ComSIS Computer Science and Information Systems
Volume 22, Number 2, April 2025

Contents

Editorial

Papers

401 **Hybrid Deployment Strategy for Software Updates to the Manufacturing Execution System Layer**
Petar Rajković, Dejan Aleksić, Dragan Janković, Aleksandar Milenković, Anđelija Đorđević

437 **Comprehensive Approach to the Design of Information Systems and Optimization of Technical Solutions according to Many Criteria**
Mikhail Tatur, Natalia Novoselova, Marina Lukashevich

459 **Research on Problem Formulations in Resource-aware Problems Across Scientific Domains and Applications**
Paweł Czarnul, Mariusz Matuszek

491 **Resource-Aware Design of an IoT Node for Use in Remote Industrial and Hazardous Areas**
Petar Rajković, Milan Vesković, Dejan Aleksić, Dragan Janković

533 **Demystifying Power and Performance Variations in GPU Systems through Microarchitectural Analysis**
Burak Topcu, Deniz Karabacak, Işıl Öz

563 **Comparison and Analysis of Software and Hardware Energy Measurement Methods for a CPU+GPU System and Selected Parallel Applications**
Grzegorz Koszczał, Mariusz Matuszek, Paweł Czarnul

591 **Manatee: A Multicore Interference Analysis Tool for Embedded SoC Evaluation**
Axel Wiedemann, Florian Haas, Sebastian Altmeyer

623 **Resource-Aware Object Detection and Recognition Using Edge AI Across the Edge-Fog Computing Continuum**
Dragan Stojanović, Stefan Sentić, Natalija Stojanović, Teodora Stamenković

649 **Energy-efficiency of software and hardware algorithms**
Maja H. Kirkeby, Thomas Krabben, Maria B. Mikkelsen, Mads Rosendahl, Mathias Larsen, Martin Sundman, Tjark Petersen, Martin Schoeberl

ISSN: 2406-1018 (Online)

Computer Science and Information Systems

Vol 22, No 2, April 2025

ComSIS

Computer Science and Information Systems

Published by ComSIS Consortium

Special Issue on Computer Systems and Resource Awareness

Volume 22, Number 2
April 2025

ComSIS is an international journal published by the ComSIS Consortium

ComSIS Consortium:

University of Belgrade:
Faculty of Organizational Science, Belgrade, Serbia
Faculty of Mathematics, Belgrade, Serbia
School of Electrical Engineering, Belgrade, Serbia
Serbian Academy of Science and Art:
Mathematical Institute, Belgrade, Serbia
Union University:
School of Computing, Belgrade, Serbia

University of Novi Sad:
Faculty of Sciences, Novi Sad, Serbia
Faculty of Technical Sciences, Novi Sad, Serbia
Technical Faculty “Mihajlo Pupin”, Zrenjanin, Serbia
University of Niš:
Faculty of Electronic Engineering, Niš, Serbia
University of Montenegro:
Faculty of Economics, Podgorica, Montenegro

ComSIS Editorial Office:
University of Novi Sad, Faculty of Sciences,
Department of Mathematics and Informatics
Trg Dositeja Obradovića 4, 21000 Novi Sad, Serbia
Phone: +381 21 458 888; **Fax:** +381 21 6350 458
www.comsis.org; Email: comsis@uns.ac.rs

Volume 22, Number 2, 2025
Novi Sad

Computer Science and Information Systems

Special Issue on Computer Systems and Resource Awareness

ISSN: 2406-1018 (Online)

The ComSIS journal is sponsored by:

Ministry of Education, Science and Technological Development of the Republic of Serbia
http://www.mpn.gov.rs/

Computer Science and Information Systems

AIMS AND SCOPE

Computer Science and Information Systems (ComSIS) is an international refereed journal, published in Serbia. The objective of ComSIS is to communicate important research and development results in the areas of computer science, software engineering, and information systems.

We publish original papers of lasting value covering both theoretical foundations of computer science and commercial, industrial, or educational aspects that provide new insights into design and implementation of software and information systems. In addition to wide-scope regular issues, ComSIS also includes special issues covering specific topics in all areas of computer science and information systems.

ComSIS publishes invited and regular papers in English. Papers that pass a strict reviewing procedure are accepted for publishing. ComSIS is published semiannually.

Indexing Information

ComSIS is covered or selected for coverage in the following:

- Science Citation Index (also known as SciSearch) and Journal Citation Reports / Science Edition by Thomson Reuters, with 2023 two-year impact factor 1.2,
- Computer Science Bibliography, University of Trier (DBLP),
- EMBASE (Elsevier),
- Scopus (Elsevier),
- Summon (Serials Solutions),
- EBSCO bibliographic databases,
- IET bibliographic database Inspec,
- FIZ Karlsruhe bibliographic database io-port,
- Index of Information Systems Journals (Deakin University, Australia),
- Directory of Open Access Journals (DOAJ),
- Google Scholar,
- Journal Bibliometric Report of the Center for Evaluation in Education and Science (CEON/CEES) in cooperation with the National Library of Serbia, for the Serbian Ministry of Education and Science,
- Serbian Citation Index (SCIndeks),
- doiSerbia.

Information for Contributors

The Editors will be pleased to receive contributions from all parts of the world. An electronic version (LaTeX), or three hard-copies of the manuscript written in English, intended for publication and prepared as described in "Manuscript Requirements" (which may be downloaded from http://www.comsis.org), along with a cover letter containing the corresponding author's details should be sent to official journal e-mail.

Criteria for Acceptance

Criteria for acceptance will be appropriateness to the field of Journal, as described in the Aims and Scope, taking into account the merit of the content and presentation. The number of pages of submitted articles is limited to 20 (using the appropriate LaTeX template).

Manuscripts will be refereed in the manner customary with scientific journals before being accepted for publication.

Copyright and Use Agreement

All authors are requested to sign the "Transfer of Copyright" agreement before the paper may be published. The copyright transfer covers the exclusive rights to reproduce and distribute the paper, including reprints, photographic reproductions, microform, electronic form, or any other reproductions of similar nature and translations. Authors are responsible for obtaining from the copyright holder permission to reproduce the paper or any part of it, for which copyright exists.

Computer Science and Information Systems

Volume 22, Number 2, April 2025

CONTENTS

Editorial

Papers

401 Hybrid Deployment Strategy for Software Updates to the Manufacturing Execution System Layer
Petar Rajković, Dejan Aleksić, Dragan Janković, Aleksandar Milenković, Anđelija Đorđević

437 Comprehensive Approach to the Design of Information Systems and Optimization of Technical Solutions according to Many Criteria
Mikhail Tatur, Natalia Novoselova, Marina Lukashevich

459 Research on Problem Formulations in Resource-aware Problems Across Scientific Domains and Applications
Paweł Czarnul, Mariusz Matuszek

491 Resource-Aware Design of an IoT Node for Use in Remote Industrial and Hazardous Areas
Petar Rajković, Milan Vesković, Dejan Aleksić, Dragan Janković

533 Demystifying Power and Performance Variations in GPU Systems through Microarchitectural Analysis
Burak Topcu, Deniz Karabacak, Işıl Öz

563 Comparison and Analysis of Software and Hardware Energy Measurement Methods for a CPU+GPU System and Selected Parallel Applications
Grzegorz Koszczał, Mariusz Matuszek, Paweł Czarnul

591 Manatee: A Multicore Interference Analysis Tool for Embedded SoC Evaluation
Axel Wiedemann, Florian Haas, Sebastian Altmeyer

623 Resource-Aware Object Detection and Recognition Using Edge AI Across the Edge-Fog Computing Continuum
Dragan Stojanović, Stefan Sentić, Natalija Stojanović, Teodora Stamenković

649 Energy-efficiency of software and hardware algorithms
Maja H. Kirkeby, Thomas Krabben, Maria B. Mikkelsen, Mads Rosendahl, Mathias Larsen, Martin Sundman, Tjark Petersen, Martin Schoeberl

Special Issue on Computer Systems and Resource Awareness [⋆] Editorial

Gordana Rakić[1], Maja H. Kirkeby[2], Padma Iyenghar[3,4], and Zoran Budimac[1]

[1] University of Novi Sad, Faculty of Sciences, Serbia
gordana.rakic@dmi.uns.ac.rs, zjb@dmi.uns.ac.rs
[2] Roskilde University, Computer Science, Denmark
kirkebym@acm.org
[3] University of Osnabrueck, Germany
[4] Innotec GmbH-TUV Austria Group, Germany
piyengha@uos.de

1. Introduction

Over the past decades, computer systems have evolved to provide high-performance and energy-efficient processing across a broad range of application domains, including mobile, embedded, data-center, and high-performance computing supporting smart, (self-)adaptive, and autonomous systems. However, realizing this potential requires system designers to be resource-aware and to manage the allocation of resources across system components under conflicting constraints. This presents a significant challenge: to understand and control trade-offs in the use of system resources-such as time, memory, energy, and data, both statically and dynamically.

Resource utilization must be considered throughout the systems' life cycle, including development, maintenance, and operation. Making informed trade-offs during specification, design, implementation, and execution requires a deep understanding of how resource usage decisions affect both individual components and the system as a whole. This is particularly important when multiple application threads execute concurrently, potentially competing for shared resources. Such awareness is crucial across sectors, including education, research, innovation, industry, and policy-making, and across all regions, for achieving sustainability in software and system engineering and usage.

The special issue ”Computer Systems and Resource Awareness” has been edited and published under the scope of COST Action CA19135[5] CERCIRAS: Connecting Education and Research Communities for an Innovative Resource-Aware Society[6], funded by the European Cooperation in Science and Technology (COST) Association[7]. COST is a

[⋆] This special issue publishes extended versions of selected papers presented at the 3 Workshops: (1) CERCIRAS 2021, (2) RAW 2022: Workshop on Resource Awareness of Systems and Society, and (3) RAW 2023: 2nd Workshop on Resource Awareness of Systems and Society.

[5] CA19135 `https://www.cost.eu/actions/CA19135/`

[6] CERCIRAS `https://www.cerciras.org`

[7] COST `https://www.cost.eu/`

funding agency for research and innovation networks. COST Actions help connect research initiatives across Europe and enable scientists to grow their ideas by sharing them with their peers. This boosts their research, career, and innovation.

CERCIRAS COST Action brings together over 400 participants representing more than 35 countries. CERCIRAS' goal is to build capacity and coordinate research to enhance resource awareness in computer and system engineering and usage. Achieving these goals requires action on two levels: (1) connecting fragmented research efforts to develop more holistic views on both problems and solutions; and (2) leveraging appropriate educational and technology assets to improve the understanding and management of resources by the academia, industry, and users of under-performing economies, ultimately contributing to economical, societal, and environmental benefits.

This special issue publishes extended versions of selected papers presented at the three editions of workshops organized under the scope of the COST Action CA19135, CERCIRAS: *Connecting Education and Research Communities for an Innovative Resource Aware Society*. The current RAW workshop series originated from the inaugural CERCIRAS Workshop, which was collocated with the CERCIRAS Open-the first Annual CERCIRAS Action meeting. The second edition of the CERCIRAS Workshop was renamed to RAW: *Workshop on Resource Awareness of Systems and Society*, and was held as part of ICT4S 2022: *ICT for Sustainability*[8]. The third edition, i.e., the 2nd workshop on Resource Awareness of Systems and Society (RAW 2023), was co-located with Euro-Par 2023: the 29th International European Conference on Parallel and Distributed Computing[9]. CERCIRAS supports the RAW workshops throughout the Action with a clear objective: to establish a discussion forum grounded in formal paper submission, peer review, and publication. The forum is intended to foster discussions on early scientific findings and their application in the area of resource-aware computing, and to extend its impact beyond the Action itself -bridging communities, disciplines, domains, and sectors while building and strengthening lasting connections.

The first article in this collection, titled "Hybrid Deployment Strategy for Software Updates to the Manufacturing Execution System Layer" by Petar Rajković et al., extends the authors' previous work on identifying common challenges in software update processes. That earlier work focused on the most effective update strategies running at the lowest (Internet of Things – IoT) and highest (Enterprise Resource Planning – ERP) levels with a special focus on energy efficiency. The further shift to the Manufacturing Execution System (MES) layer following the Industry 4.0 paradigm brought additional challenges and the contribution presented in this article. It describes a further development of the hybrid software deployment system when applied to the multi-connected levels (e.g. MES) that leads to positive results in network load distribution and significant reduction of efforts when a rollback is needed.

Mikhail Tatur et al. in "Comprehensive Approach to the Design of Information Systems and Optimization of Technical Solutions According to Many Criteria" deal with finding the optimal variant of a multiprocessor system as a multi-criteria optimization problem. The article describes several approaches to the comparative assessment of multiprocessor systems. The authors propose a ranking method to evaluate technical solutions. First, they rank individual alternatives for each optimization criterion separately,

[8] ICT4S 2022 `https://conf.researchr.org/home/ict4s-2022`
[9] Euro-Par 2023 `https://2023.euro-par.org`

and afterwards, they aggregate ordered ranked lists. The advantage of using the ranking methods is obtaining a complete technical solutions ranking based on their effectiveness according to several criteria. The article may potentially influence education, training, and practice for the domain of interest.

Another article that may have educational influence even across domains and sectors is entitled ”Research on Problem Formulations in Resource-aware Problems Across Scientific Domains and Applications”, published by Paweł Czarnul and Mariusz Matuszek. The authors focus on resource-related problems using one of the general problem and solution formulations, such as integer linear programming, greedy algorithms, dynamic programming, evolutionary algorithms, or machine learning oriented ones. The authors identify open research tasks across different domains based on identified formulations. They analyze each of these problems regarding resources to be optimized, specific optimization algorithms and domains. Finally, based on a broad review of literature, they consider the utilization of resources (time, cost, energy, human, computer, natural resources, and data) in problem formulations across domains and disciplines. Hence, the article may serve as an overview of algorithms applied on a specific domain or as an exemplification of the usage of alternative algorithms in related domains.

Petar Rajković et al. in the article entitled ”Resource-Aware Design of an IoT Node for Use in Remote Industrial and Hazardous Areas” address the challenge of designing low-power IoT nodes that integrates hardware-software co-design. The design is focused on the replacement of standard hardware components with energy-efficient ones, where the charging of batteries is also optimized. It is based on standardized components for deployment in environments where energy efficiency and autonomy are critical. Such environments are usually remote, off-grid, industrial, and hazardous. On top of such hardware configuration, software components offer over-the-air updates and reconfiguration. The node is integrated into a cloud-based digital twin with centralized control over the complete operation cycle. Results show that the proposed node architecture reduces energy to 50%, while in some cases, consumption is reduced by up to one-tenth compared to starting designs. Finally, the article delivers a set of design recommendations for adapting the standard components for harsh environments.

”Demystifying Power and Performance Variations in GPU Systems through Microarchitectural Analysis”, written by Burak Topcu et al. describes a novel memory performance and power monitoring tool for GPU programs. The tool is called GPPRMon and performs a systematic metric collection visualizes them and guides power and performance analysis for target executions based on simulation. The tool gathers metrics that reflect system and memory-related microarchitectural characteristics by monitoring individual instructions and reports collected performance and power consumption information, all dynamically at runtime. The mentioned visualization provides spatial and temporal views of the execution. Authors use the described tool to demonstrate performance and power analysis on GPU benchmark suites, focusing on memory-bound graph applications and resource-critical embedded programs. The results show the potential for application of the tool in memory-bound kernel identification, performance bottleneck analysis of a memory-intensive workload, performance power evaluation of an embedded application, and the impact of input size on the memory structures of an embedded system.

”Comparison and Analysis of Software and Hardware Energy Measurement Methods for a CPU+GPU System and Selected Parallel Applications” by Grzegorz Koszczał et al.

is an extension of the previous work on power capping in optimization of performance-energy metrics of deep neural networks training workloads. They use a professional power meter Yokogawa WT-310E, Intel RAPL and Nvidia NVML interfaces, to observe power consumption of multi-GPU and multi-CPU configurations, such as selected kernels from NAS Parallel Benchmarks for CPUs and GPUs and Horovod-Python Xception deep neural network training using several GPUs. The authors collect, compare and discuss results collected by both power measurement methods performed using: (1) 2 Intel Xeon CPUs and 8 Nvidia Quadro RTX 6000 GPUs, and (2) 2 Intel Xeon CPUs and 4 Nvidia Quadro RTX 5000 GPUs. Finally, the authors compare power consumption between hardware and software interfaces for CPU, GPU and mixed CPU+GPU workload configurations, using 1-40 threads for the CPUs and 1-8 GPUs.

Axel Wiedemann et al. in ”Manatee: A Multicore Interference Analysis Tool for Embedded SoC Evaluation” present the improvement of the RISC-V hardware development framework Chipyard by a tool which gives it awareness of alternation for shared resources during the design and development process of System on Chips. The tool moves the agile development focus of the framework to the effects of changes that are related to the shared resource alternation. The focus is moved by adding the capabilities for quick feedback upon a change regarding its effects on the shared resource. The prototype of the tool is tested and evaluated on a use case.

In the article titled ”Resource-Aware Object Detection and Recognition Using Edge AI Across the Edge-Fog Computing Continuum”, Dragan Stojanović et al. address the challenges of deploying deep neural networks for edge intelligence and traffic object detection and recognition on a video captured by edge device cameras. The authors mainly aim to analyse resource consumption and the achievement of resource awareness. They optimize computational resources across diverse edge devices within the edge-fog computing continuum while maintaining high object detection and recognition accuracy. The authors propose a method to exploit the edge-to-fog paradigm to distribute the inference workload across multiple tiers of the distributed system architecture. Implementation of the proposed method in the context of edge-fog-related solutions is evaluated based on several use cases. Results show improved accuracy of recognition and resource utilization, as well as adaptability of the system to dynamic traffic scenarios, ensuring real-time recognition performance even in challenging environments.

Finally, the article on ”Energy-efficiency of Software and Hardware Algorithms” by Maja H. Kirkeby et al. concludes this Special Issue. The authors compare the energy efficiency of algorithm implementations in hardware and software, with the aim of establishing a fair basis for comparison between the two. Specifically, the study investigates conditions under which time and energy efficiency diverge. To this end, they analyze implementations of the Heapsort sorting algorithm and Dijkstra’s path-finding algorithm written in C for the Raspberry Pi, and in Chisel for deployment on an FPGA. The results contribute to a deeper understanding of the trade-offs between performance and energy efficiency across different computational platforms.

cost EUROPEAN COOPERATION IN SCIENCE & TECHNOLOGY

Funded by the European Union

CERCIRAS

Computer Science and Information Systems 22(2):401–435 https://doi.org/10.2298/CSIS240807027R

Hybrid Deployment Strategy for Software Updates to the Manufacturing Execution System Layer*

Petar Rajković1, Dejan Aleksić2, Dragan Janković1, Aleksandar Milenković1, and Anđelija Đorđević1

1 University of Niš, Faculty of Electronic Engineering,
Aleksandra Medvedeva 4, 18104 Niš, Serbia
{petar.rajkovic, dragan.jankovic, aleksandar.milenkovic, andjelija.djordjevic}@elfak.ni.ac.rs
2 University of Niš, Faculty of Science and Mathematics, Department of Physics,
Višegradska 33, PO BOX 224, 18106 Niš Serbia
alexa@pmf.ni.ac.rs

Abstract. Complex industrial systems consist of many heterogeneous devices running different hardware and software in a connected, layer-organized environment. Since all these software instances must be updated occasionally, and since they could affect the layers under and above, the definition of deployment strategies that will reduce downtime is necessary. In previous work, we focused on identifying common problems in software update processes and concentrated on the most effective update strategies running at the lowest (Internet of Things – IoT) and highest (Enterprise Resource Planning – ERP) levels. The result was a set of recommendations and strategies that should help minimize network utilization and processing resources and make the process as energy-efficient as possible. After that, the core effort of the research is shifted toward the Manufacturing Execution System (MES) layer – the layer that brings the higher complexity, both in terms of connectivity and software complexity. Following the actual Industry 4.0 paradigm, the software in the MES layer becomes even more critical since it is expected to integrate a whole new set of responsibilities previously belonging to various levels or external solutions. To facilitate further requests, deployment strategies are reevaluated and enriched with innovative approaches such as A/B testing and the separate update service. This paper shows the possible further development of the hybrid software deployment system when applied to the multiconnected levels, such as the MES. The adaptation shows positive results regarding the network load distribution and significant effort reduction when a rollback is needed.

Keywords: computer science, information systems, Word, typesetting.

1. Introduction

Complex industrial systems represent an exciting conglomerate of various technical solutions. Knowledge from different engineering sciences is needed to solve the

* This manuscript is an extended version of the paper published in the proceedings of the CERCIRAS 2021 workshop.

challenges from process modeling, signal collecting, and processing through plant layout design to raw materials and finished goods transportation, distribution, and storage. Nowadays, all these aspects are supported by adequate software. Due to significant differences between diverse aspects of the organization in the industrial facility, the complete structure is divided into standardized layers. The standard ISA95 [1] defines in detail how to split the industrial system organization and what the responsibility of each layer is. Following the standard, the information technology (IT) subsystem in the industrial environment consists of many heterogeneous devices running different pieces of software in a connected and layer-organized environment [2] (Fig. 1). Starting from the sensor and actuator layer connected with microcontrollers (in our work, we will reference it as the IoT layer) [3], through the Edge layer [4][5], via SCADA [6] and manufacturing execution systems (MES) [7] to enterprise resource planning (ERP) [8], all pieces of equipment run the software that needs to be updated occasionally.

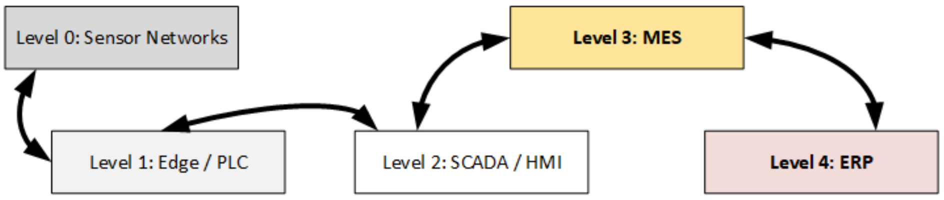


Fig. 1. ISA95 levels according to the industrial standard [1]

Software update, as a process, is an activity that is considered highly problematic in the industrial environment. From the point of view of the process engineer, it should either happen never or only in predefined maintenance slots. It came from the experience with the previous deployment methods, where intensive planning must be done, and some areas of the industrial facility will be disconnected for a more extended period. If deployment needs to be reverted or reconfigured, the problem will be even more significant.

In our previous work, we have been focused on the deployment of the software in the lowest (the IoT level [9]) and the highest (the ERP level [10]) levels of the system. From the connectivity point of view, these two layers have been of minor operation complexity since they maintain connectivity only to nearby levels. IoT nodes are usually connected only to Edge computers, while the ERP communicates with MES. The main difference between them is the requirements regarding the volume of the needed resources. In the IoT area, resource shortages are faced in every aspect of work.

This paper represents the direct extension of the work published in the CERCIRAS 2021 workshop. The definition of the testing environment and the default deployment strategies used for IoT nodes were the starting point and thus included in this work. This paper describes the usage of the concepts of the software update approach for a single node with limited storage space and expands then further on the application at the MES level.

As mentioned, any deployment strategy must consider energy consumption, storage space, and processing power. Such an environment requires carefully defined deployment methods and, even more importantly, backup and restore strategies in case of unsuccessful deployments. The next step was to generalize the approach described for

IoT nodes and apply it to the ERP layer [10]. In that sense, this paper could be seen as the further continuation of the work we described in [10]. Since ERP layer software was built with more advanced software tools, it offers more possibilities for defining the update strategy. In that sense, the different software deployment methods were analyzed, and a set of routines that should improve deployment scenarios was proposed and evaluated. Deployment strategies, defined in [10], were the next step in our deployment and were thus used as another starting point in our work. The explained use of advanced strategies was another building block to define routines for the MES software. The software at the ERP level shares the complexity, technology stack, and implementation approaches with MES, which was of significant value for this work.

The next goal is to apply the proposed deployment routines to the MES layer. MES is the layer that brings the higher complexity into the design, both in terms of connectivity and software complexity. Following the actual Industry 4.0 paradigm, the software in the MES layer becomes even more critical since it is expected to integrate a whole new set of responsibilities previously belonging to various levels or external solutions. Deployment strategies are reevaluated and enriched with innovative approaches to facilitate further requests. For example, the MES server could be connected to SCADA on one side and to the ERP on another. In contrast, the clients could be connected directly to measuring devices in Edge or IoT to register and visualize different measurements.

In this situation, downtime during the update needs to be evaluated through multiple sides to ensure proper reconnection and operation continuation from various sides. Also, one must remember that with new requirements under Industry 4.0, the MES software should offer new functionalities that often come without full specification and where multiple versions must be simultaneously evaluated. This paper shows the results of the research that had the following research tasks:

- Test and adapt the deployment strategies suggested in [9] and [10] and try to use them both for server and client components of the MES level.
- Focus to reduce network load on the MES server side.
- Organize deployment to stop the erroneous deployment as soon as possible.
- Integrate the process of the practical test of new functionalities when the customer must choose between multiple solutions.

This research relies on our previous work, primarily described in [9] and [10], and represents its continuation and improvement.

2. Background – Industry 4.0 Paradigm and Existing MES Systems

MES and Industry 4.0 are critical components of the modern manufacturing landscape. They aim to integrate technology and data to optimize production processes, improve efficiency, and drive innovation in many new ways. Industry 4.0, the Fourth Industrial Revolution, represents a radical shift in manufacturing practices.

It involves the digitization and the use of advanced scheduling and execution algorithms in manufacturing processes, moving away from mass production towards customized production that caters to individual customer requirements (Fig. 2). This means that a portion of the planning and scheduling will be moved from ERP to the MES level. Next, MES plays a crucial role in Industry 4.0 by providing real-time visibility, control, and intelligence across the entire product life cycle value chain. It should allow for seamless communication, analysis, and data utilization to drive intelligent actions in the physical world. This means that the connection from MES will not only go to the SCADA layer but also directly to Edge, IoT, and sensor networks in some cases.

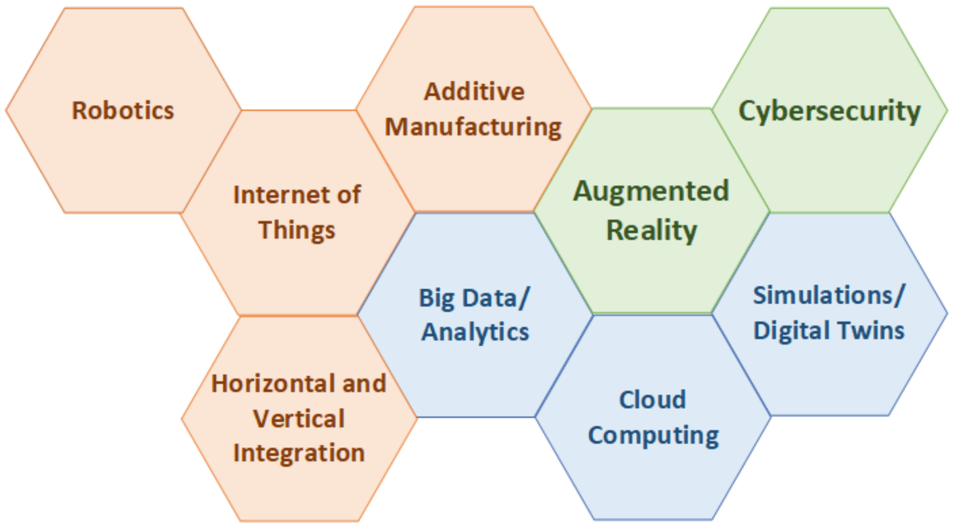


Fig. 2. Main elements of Industry 4.0

With full rights, the new generation of MES and Industry 4.0 is expected to enable organizations to harness the power of digital technologies and intelligent, connected systems to revolutionize their manufacturing processes. They would allow organizations to optimize production processes, improve efficiency, and drive innovation by leveraging robotics, analytics, artificial intelligence, nanotechnology, the Internet of Things, and cloud computing. These technologies enable organizations to automate tasks, analyze data for actionable insights, and connect various parts of the production process for seamless coordination and optimization. At the same time, it is, more than ever, expected that software runs with the lowest possible downtime and that all activities run as smoothly as possible.

While previous work focused on single-connection levels, like IoT and ERP, this paper will evaluate the application and extension of the existing set of recommendations for software at the MES level. MES-level software significantly differs from those running in IoT nodes but is closer to ERP systems. First, MES systems usually follow service-oriented architecture (SOA) with various clients.

These software instances run on servers or in the workstation, with significant processing power and memory storage compared to IoT nodes. It looks like the MES systems run in an environment where resources are not the problem, but it is not quite like that. Depending on the configuration and the set of required operations, MES clients could weigh up to a few hundred megabytes. It depends, of course, on the implementation technology and other dependencies. Still, if they are implemented as the thick client, the usual user requirement, their update process could employ significant network traffic.

Compared to ERP software, MES runs fewer complex algorithms, but it connects more extended software and services and runs in significantly more numbers and variants of clients. It is also essential to state that with the current technology demands fueled by the Industry 4.0 initiative, the importance of the MES system rose. Nowadays, MES is often required to provide many functionalities native to other systems. The MES should now support continuing different reporting, overall equipment effectiveness tracking (OEE), Andon boards, deeper integration with ERP systems and SCADAs, and ending various synchronizations with warehouse, packaging, and other systems.

3. Related Work

The existing literature offers various deployment strategies, evaluations, and recommendations. In most cases, the existing research covers software that runs in layers such as MES and ERP. Besides, it has been constructive for our current scope of research, but it was a bit misleading when one tends to define the close-to-universal strategies and approaches. These higher layers deal with clients transferring significant data and executing numerous transactions. When defining development strategies for lower levels, the standard approaches from the literature are not directly implementable due to their unique limitations.

The most critical points for resource management at lower levels are storage capacity and data traffic through connecting networks. The overall effect is not the same on all layers [15]. MES runs in a shop floor environment on devices with processing power similar to standard computers.

The storage space is not a critical requirement for devices running MES or ERP software, but they are usually connected to their server using the wireless network. The wireless networks in the industrial environment could experience different disruptions because of operating nearby machines generating high-frequency harmonics and other security threats [16]. Data package verification and consistency are critical for MES and ERP client nodes. When deploying a new software version to some device, an update package of significantly higher volume than usual data traffic needs to be distributed via a network, verified, and stored on the destination device. The old version needs to be backup in case of rollback [17] [18]. Next, the Edge layer's primary mission is to collect all the data from sensor networks and pass it to the MES. In this case, the proper buffer implementation ensures smooth software upgrades.

All the mentioned layers are highly heterogeneous, with different pieces of hardware running the software instances with diverse categories of software. Overall, in the complete industrial system, the type of used devices, their number, and the amount of transferred data (per device) could be between 1kB and 1GB. To make the complete process more demanding, the devices sometimes do not have enough memory to store two software versions; thus, they would require backup in a different location. This leads to the situation that sometimes it is nearly impossible to upgrade with no downtime or at least with very low downtime [19].

As with every process, a software update could fail for numerous reasons. In that case, a complete deployment approach or deployment system must provide the possibility to roll back to the previous version [20]. The rollback will then take more resources and worsen the situation, so we need to ensure that system governance successfully goes through the process [21].

To reduce the impact of the mentioned problems and potential system downtime, we aimed to define a more general approach that could be configured to use the combination of blue-green [22] and canary deployment [23] styles in combination with both shared and local backups [24]. This approach looks promising at the IoT level. The approach was tested in a production environment, and the results were published in [9] and [10].

Working on a general set of recommendations [9] [10], we conclude that regardless of the type of software and the operating level, the blue/green approach could be effectively used at any node (Table 1). New components used to build IoT nodes increased memory and processing power, so keeping two versions simultaneously would probably not be a problem. The blue/green approach, per se, could be improved with additional techniques such as buffers and backup nodes [9] [10]. For example, at all levels, a blue/green approach supported by the dark mode with feature flags could be used for server node deployment. This will give flexibility and security; newly developed features could be gradually turned on until the complete server update is reached. For clients, blue/green is the primary choice, which could be enriched with buffers and feature flags if the resource pool and used implementation technology allow.

Table 1. Elements of the deployment strategy used in various levels (BG – blue/green, DF – dark mode with feature flags, CS – canary with sentinel node, CB – canary with backup node, IB – intermediate buffer, (XX) - optionally) (as suggested in [10])

Level	Server	Client network	Single Client
Levels 0 and 1 (sensor network)	BG + (DF)	CB + IB	(BG) + IB
Level 2 (IoT nodes and Edge computers)	BG + DF	CB / CS + IB	BG + IB
Level 3 (MES)	BG + DF	CS / CB	BG + DF + IB
Level 4 (ERP)	BG + DF	CS	BG + DF + IB

The level of downtime reduction is significantly reduced in this scenario compared to standard approaches such as recreate deployment and rolling deployment [27]. In the recreate deployment, the previous version of the software is shut down, and the new one starts after the old one has been stopped. Rolling deployment is applicable for complex systems with multiple servers. It is based on the recreate deployment but applies to various services. The downtime is exceptionally low, but the length of an upgrade process depends on the number of servers/nodes in an array, and it could take considerable time. The proposed deployment strategy will improve overall deployment time even more in the case of the rolling strategy since the blue/green switch could be done in the close period; there is no need to wait until all the servers are updated in the sequence.

The new request that does not fit into the proposed framework is to have the possibility to support simultaneous evaluation of different versions of functionality.

Besides, it could be done through the feature flags, but it will eventually require more consolidation and stabilization work. The A/B testing deployment approach is included to address such requests. This approach is used on the client side to improve the development and test phase and provide the possibility for limited testing in the production environment. This approach aims to offer different functionalities to some clients and then evaluate the user's reaction and acceptance. The update is usually done in a few groups of varying sentinel nodes.

Recreating and rolling deployment are crucial concepts in software development and operations at the MES level [28]. Recreating refers to rebuilding a software system or environment from scratch, often to resolve issues or update components. On the other hand, rolling deployment involves deploying new software versions in a gradual and controlled manner, allowing for continuous delivery and minimizing downtime. Integration with other systems and services traditionally occurs at the end of a development life cycle, but rapidly developed applications are integrated almost immediately. Testing occurs during every iteration, enabling stakeholders to quickly identify and discuss errors, code vulnerabilities, or complications and immediately resolve them without impacting the development progress. As stated in [29], “integration with other systems and services traditionally occurs at the end of a development life cycle, but rapidly developed applications are integrated almost immediately”. This iterative approach to development and testing is a crucial aspect of recreating and rolling deployment methodologies.

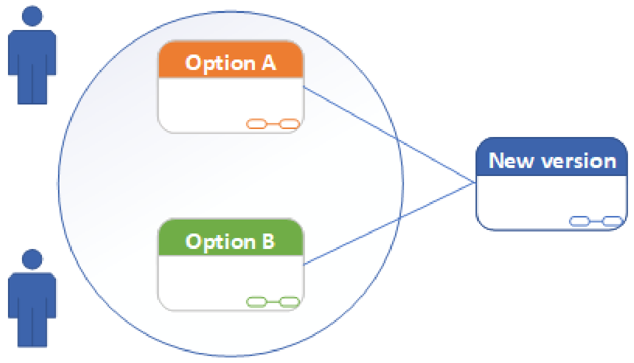


Fig. 3. A/B testing deployment

A/B deployment (Fig. 3) strategy has become increasingly popular in various fields, including technology, marketing, and product development. This strategy involves testing a product or service's versions, A and B, to see which performs better [25]. The first source highlights the importance of product or service innovations in engaging customers and improving performance. It suggests that the market development strategy, which focuses on pursuing additional market segments or geographical regions, can increase sales but also comes with more risk. The second source discusses different methods for gaining market share, including product development and market development [26]. In the Industry 4.0 era, the use of A/B testing deployment is a comparable advantage within the installations of MES. The installation supports A/B testing and easy transition to the new version, which is considered more advanced and customizable. The A/B testing is widely popular with deployment based on container technologies, such as Kubernetes [30], since they involve end-users in decision-making over the new version of the software.

4. Testing Environment

As it has been known, the update process comes with the risk of diverse potential failures that could leave parts of the system unresponsive, running with unpredictable behavior, or emitting erroneous data. For this reason, the update process must be executed in a highly controllable environment that allows easy and efficient rollbacks in case a flawed deployment is detected. As stated before, all software components in the industrial system are usually organized in layers. Layers exchange data with each other using different software protocols. The mentioned facts make the overall software update process a bit more complex than within a standard information system environment, and every error could lead to serious domino effects [11] [12]. Updating software in one layer could impact the targeted device and other devices in the same and different layers. For example, the update performed on the device running at the MES level could affect software instances running in other layers.

The additional limitation point is the expectation for the highest possible performance and the requirement that software run using as few resources as possible. The complete system must have a high degree of resource awareness, and both storage space and network bandwidth usage must be carefully planned during the update process in order not to reduce the execution of the running components significantly [13][14]. For this reason, the resolute digital twin is used for testing.

The digital twin (Fig. 4) is created partly in the laboratory environment and partly in the cloud to simulate different connectivity scenarios and have an overview of worse-case scenarios regarding latency and execution. The emulated hardware in a digital twin is set to the lowest acceptable resource level, which should simulate worse execution conditions than those in the production environment. The testing digital twin is introduced while implementing the one-of-the-kind production system [32]. As the demo factory, the plant producing doors and windows is set.

Such a production facility is used for demonstrating since it combines all diverse kinds of production and needs multiple sensors and precise mechanical units to be integrated. On the MES, the level needs several diverse types of clients and services. The digital twin environment used for testing was described in [10] and improved to support more complex environments. Previous research focused either on IoT nodes, which were entirely configured in the local network, or on ERP clients, which were all the same and ran only in the cloud. The IoT level in the digital twin consists of 100 nodes connected to simulated instances of sensors and actuators. Each IoT contains a different number of sensors and actuators, which count within the node and could be anything between a few and 1,000. The count of 100 gives enough flexibility and complexity to perform testing in the development phase.

The digital twin, an exact mirror replica of the industrial facility environment, could be created for the production phase. In the default model, following the ISA95 model, sensors are connected to IoT nodes. Especially after the Industry 4.0 concept brought new requirements for MES systems, a direct connection between MES clients and measuring sensors could be established, too.

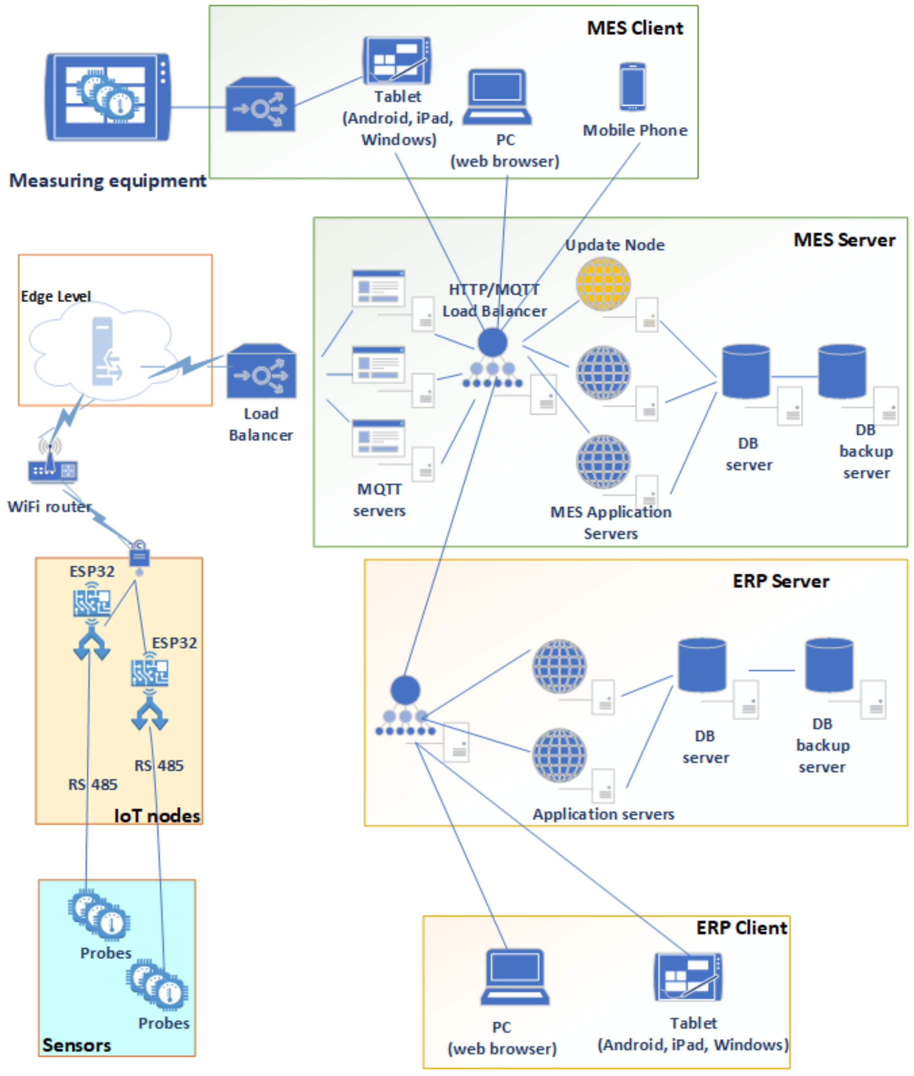


Fig. 4. The composition of the examined system containing all levels of the ISA95 model

Sensors within one IoT node could be different, and all could run various software. Sensors could be active either constantly or just for predefined periods. They could collect heterogeneous data with varying sample rates during their operation time. All these facts make the IoT level very dynamic from the operational point of view. They could increase the probability that the complete node went out of a stable state in case of problematic deployments.

The available memory space is usually between 1 and 5 MB per device, which is enough for the necessary software. The nodes in the IoT layer are connected using various methods, ranging from cable network connectors to LoRaWAN, which creates an inconsistent environment in terms of connection speed and quality. The most complex situation is with LoRa-connected devices since their bandwidth could be only 10-20 kbps.

IoT node layers are further connected to Edge computers or Edge nodes. Edge nodes communicate between the shop floor and hazardous areas on one side and higher levels, such as MES and enterprise resource planning (ERP), on the other. Edge nodes are devices based on Raspberry Pi or similar base sets and are usually connected by a Wireless network with an effective network speed of around 20 Mbps. Their space requirements are around 30 MB per node. There were 10 of these nodes in our test environment. To support testing, the mesh of 10 Edge computers is modeled in digital twin. Each of them is set to collect data from 10 IoT nodes.

From the resource awareness point of view, software components on MES and ERP levels are easier to manage. They run on desktop/laptop computers with enough processing power, disk space, and bandwidth, but resource planning is inevitable even with them. In our test environment, we used 200 MES clients connected to 4 MES servers (two load-balancing and two redundant, with the possibility to change the configuration) and 30 ERP clients connected to the Microsoft Dynamics server. All the clients at this level are a few hundred megabytes in volume and are located under a gigabyte network.

Table 2. Different MES clients and their functionalities

MES Client Type	Connection within MES Level	Connection to other levels/services
Administrative	Server	ERP
Operation	Buffer, Server	Edge
Configuration	Server	ERP, External cloud services
Management	Server, Operation clients	Reporting
Measurement	Server	IoT, Edge

Different connectivity and execution actions support different types of MES clients (Table 2). Administrative clients perform operations related to the ERP level. They are responsible for synchronizing operations definitions, catalog data, material definitions, and other master data needed to properly exchange data between MES and ERP.

The operation client has a connection to the execution buffer on the MES side and to the Edge level. The execution buffer is an optional implementation that allows clients to continue to run when the server is offline. It contains a buffer filled with tasks that must be executed in the workstation and collects data generated during production. Once the connection is reestablished, the data flow will resume, and the server-side upgrade will have the lowest possible impact on the clients.

The configuration client is described in detail in [32]. It is used to define new products and eventually upload these data to cloud services and ERP. The management client acts as a synchronization node between ERP and operation clients. It is responsible for downloading production orders from ERP and uploading collected status change data measurements, etc. Ultimately, the measurement client will provide the interface for material registration and integration with IoT nodes such as sensors and other measurement devices.

5. Transition of Deployment Strategy from IoT to MES node

The software update process for IoT nodes and sensor/actuator devices running in a production environment is particularly sensitive. In industrial automation, sensors and actuators emerge as fundamental components that underpin efficient, safe, and precise operations. These unassuming devices are pivotal in monitoring, controlling, and optimizing various processes across diverse industries. The update of such small components requires detailed planning before an update. Thorough planning is needed because they are, on the one hand, tiny both in size and capacity and on the other hand, they are running in a hazardous environment where the only possible connection is relatively slow LoRa networks with no wiring possible and limited physical access, (Fig. 5). If some physical intervention is needed, the stoppage of the complete industrial process is often a requirement.

Besides the slow network, the low-performance hardware is one additional potential problem. This fact could result in an unacceptable long update process, which could move the targeted device off the system for an extended period. The last, but not the least important, is the energy consumption problem. Software updates are an activity that requires significantly more energy than regular data collection and data transmission processes. Thus, this process must be planned for when the battery is charged to the highest possible level and when the eventual rollback will not drain the battery.

At first sight, it looks like there are no common issues or problems between IoT and MES clients. MES clients have fewer limitations, especially in processing power and storage capacity. This statement suggests that one can assume that any kind of deployment strategy is convenient for MES clients. It could be said this from a strictly technical perspective, but when including different business requirements, it turned out that deployment at the MES level must be carefully designed, too. Furthermore, the main building blocks for both clients are similar (Fig. 5). In both client types, regardless of different implementation technologies, Communication, data collection, and the processing block could suffer from the same problem. The problems with the low energy level are related to IoT, while the MES clients could suffer from synchronization and compatibility problems.

Noticing this, we realize that the deployment strategy defined for IoT nodes could apply to MES clients and be enriched with the experience through the ERP client deployment project as presented in Table 1. Blue/green deployment could be used if the destination node has enough storage space. The difference would be in the specific implementation technology, but the concept will remain the same. Additionally, an intermediate buffer, defined at the IoT node level, could be safely applied to the MES level. The MES nodes implementation is based on the concept from the IoT level and then enriched with additional features that will bring even further benefits to the MES level.

Traditionally, the MES nodes usually used some of the classic deployment methods—recreate or rolling deployments. Such an approach has been acceptable in recent years. Still, due to the manufacturing shift towards Industry 4.0, users started looking at the re-installation process connected with downtime as a problem. In the case of rolling-like deployment, the issue relates to a long waiting period until the new version becomes fully available.

Furthermore, such an approach would require an IT assistant in the facility, ready to help, run an installer, or perform some similar support activity. Since this was not acceptable anymore, we aimed for an approach already applied in IoT nodes and for its transition to MES-level software.

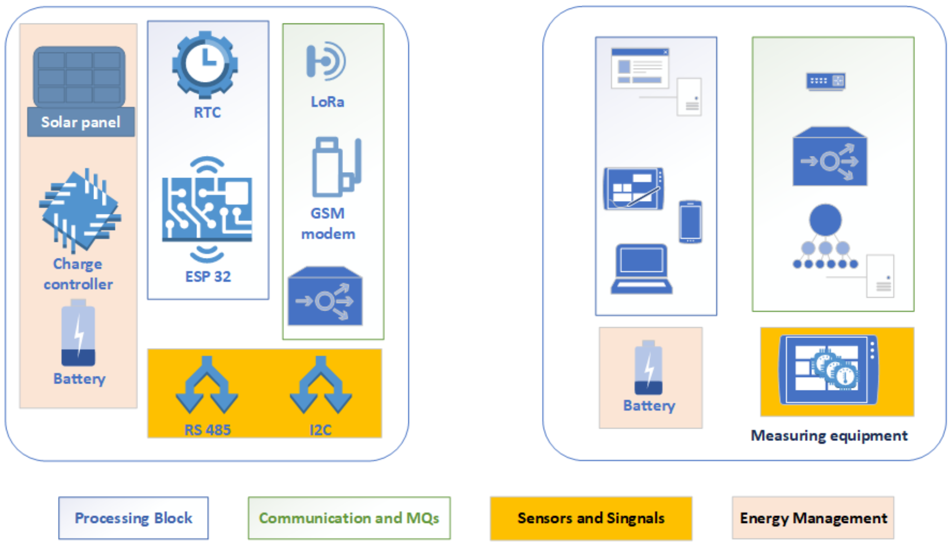


Fig. 5. Comparison of building blocks of IOT (left) and MES (right) client nodes

5.1. Software Update Approach for IoT and MES Nodes

Looking at the single IoT node, our choice for a software update is a semaphore-based green/blue approach (Fig. 6). This approach is possible with devices storing at least two software versions simultaneously. In this case, the critical points are typically low bandwidth and possibly low battery levels. The approaches to solving these two problems are elaborated further in [34].

The problems with applying such an approach at the MES level resemble the IoT level. First, data storage limitation is not, per se, the main issue, but the device could run into such a problem when the access rights for the installer are not managed correctly. The issues with access rights are not present in the IoT node since the vendor is responsible for hardware and software. At the MES level, the software is installed, in most cases, on the customer's equipment, for which the IT security and management team is responsible for maintenance.

As mentioned, the problem with low space could appear at the MES level if the installer has no delete rights for older versions. Since the MES clients could come with a few hundred megabytes of installed software and generate large log files, the issue with the space could arise if the delete and backup processes are not managed correctly.

Next, the installation could also create bandwidth problems if not appropriately managed. For example, in a factory with 200 workstations, each would require an MES client installed. In some cases, more MES clients could be launched on the same machine. At least 200 clients will require an update when an updated version is detected. If distributed from a single spot, as often chosen, the update process could easily make a bottleneck in the network. Furthermore, the MES client will maintain a connection to more layers in the ISA95 structure, which could cause further synchronization problems. For comparison, nodes at the ERP level, closely elaborated in [10], do not have connections to another system, which makes them much easier to handle.

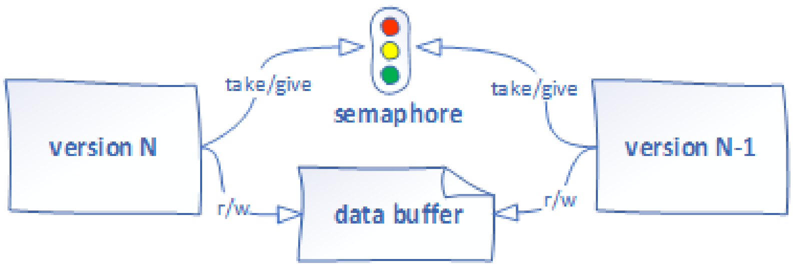


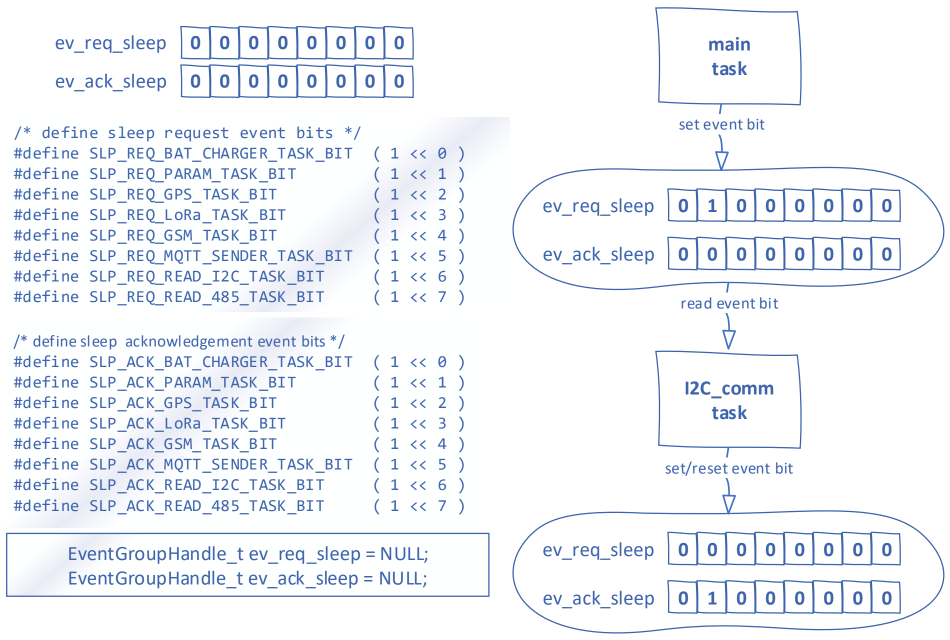
Fig. 6. Semaphore-based blue-green deployment strategy used for IoT nodes [10]

Coming back from IoT nodes, the base for the deployment approach is a blue/green strategy. This is the backbone of our update system. It is easy to be implemented in any technology. The main idea behind the blue/green strategy is to ensure that the target device always keeps at least two software versions – actual running (version N-1) and previously verified (version N-2). To reduce the data loss during the switchover, the node setup is completed by a message queue. Message queue collects data from sensors, and data are removed from the queue after being processed. The queue could be implemented as an independent entity to continue collecting data during the switchover.

The update process starts by replacing version N-2 with the new version N. At that moment, version N-1 is still active, and the device runs uninterrupted. During that period, the device experiences higher-than-average network traffic and battery use. Once version N – 2 is deleted and version N is uploaded and verified, the switchover could start. The device begins operating version N, but its communication points remain inactive. When version N is fully up and running, the semaphore opens communication to version N and stops version N-1.

In that case, there is almost no operation downtime, and the complete update process is seamless for the customer (Fig. 7). In a well-orchestrated process, data loss during the switchover can be effectively mitigated. In the worst-case scenario, only signals received during the switchover—typically lasting several seconds—may be lost and left unprocessed. The switchover is seamlessly executed for IoT nodes by transitioning to sleep mode. Since sleep modes are an integral part of processing, facilitated by a dedicated core, transitioning to and from sleep mode is considered a native operation for IoT nodes.

In many cases, this approach will also be fully applicable to MES nodes. Unfortunately, not always. Two central problems appeared here with MES clients. First, as mentioned before, the older version (N – 1) will not be deleted in case of a lack of privilege. If not managed properly, this will cause a problem with the space on the destination node. The next problem is the switchover phase. MES clients are much larger pieces of software with a powerful GUI that maintains integration with different services on the MES level and even to different Edge, SCADA, and IoT devices. The proper switchover would require replacing the client version and reestablishing a connection to other connected instances (Fig. 8). This makes the buffering system even more important here than at other levels.


ev_req_sleep 0 0 0 0 0 0 0 0
ev_ack_sleep 0 0 0 0 0 0 0 0
main task
set event bit
ev_req_sleep 0 1 0 0 0 0 0 0
ev_ack_sleep 0 0 0 0 0 0 0 0
read event bit
I2C_comm task
set/reset event bit
ev_req_sleep 0 0 0 0 0 0 0 0
ev_ack_sleep 0 1 0 0 0 0 0 0

```
/* define sleep request event bits */
#define SLP_REQ_BAT_CHARGER_TASK_BIT  ( 1 << 0 )
#define SLP_REQ_PARAM_TASK_BIT        ( 1 << 1 )
#define SLP_REQ_GPS_TASK_BIT          ( 1 << 2 )
#define SLP_REQ_LoRa_TASK_BIT         ( 1 << 3 )
#define SLP_REQ_GSM_TASK_BIT          ( 1 << 4 )
#define SLP_REQ_MQTT_SENDER_TASK_BIT  ( 1 << 5 )
#define SLP_REQ_READ_I2C_TASK_BIT     ( 1 << 6 )
#define SLP_REQ_READ_485_TASK_BIT     ( 1 << 7 )

/* define sleep acknowledgement event bits */
#define SLP_ACK_BAT_CHARGER_TASK_BIT  ( 1 << 0 )
#define SLP_ACK_PARAM_TASK_BIT        ( 1 << 1 )
#define SLP_ACK_GPS_TASK_BIT          ( 1 << 2 )
#define SLP_ACK_LoRa_TASK_BIT         ( 1 << 3 )
#define SLP_ACK_GSM_TASK_BIT          ( 1 << 4 )
#define SLP_ACK_MQTT_SENDER_TASK_BIT  ( 1 << 5 )
#define SLP_ACK_READ_I2C_TASK_BIT     ( 1 << 6 )
#define SLP_ACK_READ_485_TASK_BIT     ( 1 << 7 )

EventGroupHandle_t ev_req_sleep = NULL;
EventGroupHandle_t ev_ack_sleep = NULL;
```

Fig. 7. Software update sequence with the sleeping sequence

Blue/green is not a favorable solution; it is only for successful updates. It proves its value when the update fails. In that case, blue/green offers an effortless way to switch back to the previous (valid and proven) version N—1. Furthermore, such a rollback will not require additional data traffic, which is desirable in any scenario and level. Once the error is solved, version N could be replaced with the next update.

The blue/green setup supports both full and partial version updates. In case of a partial version update, the new version will be generated when the copy of N-1 gets merged with new libraries and configuration files. The partial approach is faster and brings a lower network load. It is helpful for MES-level clients, but it is even more suitable for devices with more processing power on the IoT level. The easiest way to spot them at the IoT level is to check if they use GSM modems and LoRa adapters. In brief, partial deployment is more efficient for more complex software components.

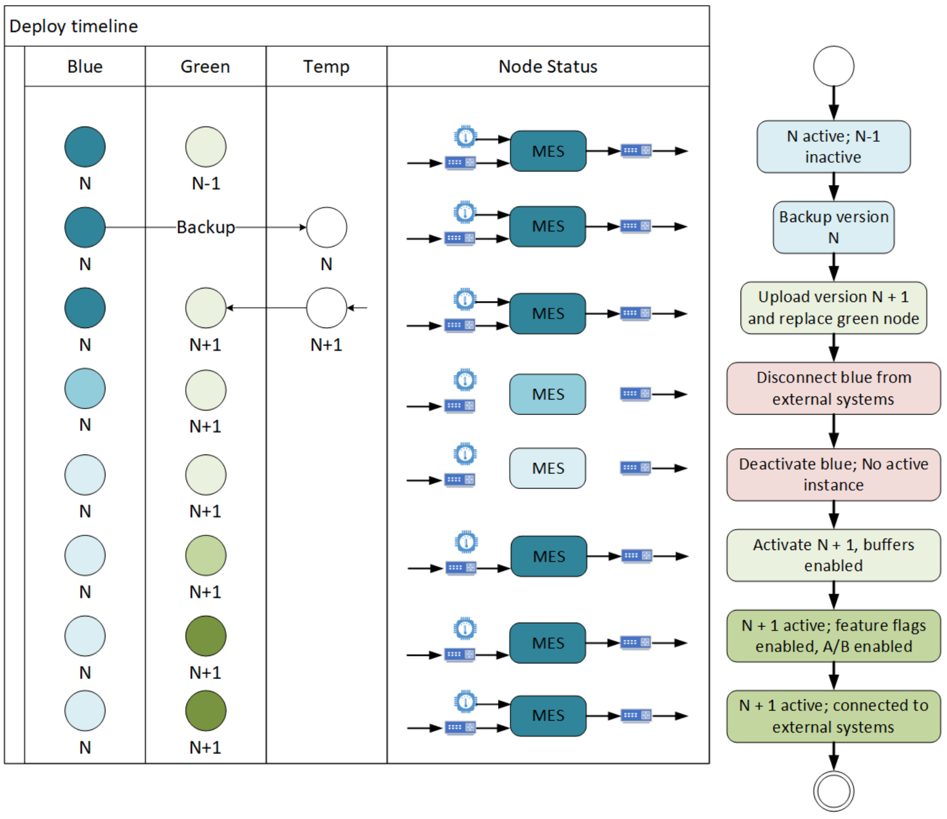

Deploy timeline
Blue
Green
Temp
Node Status
N
N-1
Backup
N
N+1
MES
N active; N-1 inactive
Backup version N
Upload version N + 1 and replace green node
Disconnect blue from external systems
Deactivate blue; No active instance
Activate N + 1, buffers enabled
N + 1 active; feature flags enabled, A/B enabled
N + 1 active; connected to external systems

Fig. 8. Software update sequence for MES client (expanded from [10])

This approach will not solve every deployment problem. In some cases, it could be inefficient or even useless. In case of a partial update, it could happen that the deployment package did not come with all necessary dependencies. Then, the update will fail, leading to additional data transfer and new version creation.

Next, the new version might be larger than the available space, even after deleting version N-1. In this situation, the blue/green approach cannot give positive results, and the update will fail. This would lead to the request for additional intervention and, in the best case, reducing the deployment to recreation mode.

Since the software is connected to services and other running instances on various levels, their interface might change occasionally. Or even buffer service needs to be updated. Blue/green will not help or solve the problem if this happens. Such updates then need to be implemented during planned downtime and meticulously organized to follow all necessary steps in the required order.

The last but not the least essential problem is when the device runs out of power during the update process. It could happen to any device, but those running on battery are more prone to this problem. The mentioned problem is not typical for MES nodes. They are connected to standard LAN/WLAN or Profibus network and are usually connected to the continuous power supply. If they lose the power during the update, they will continue to run the N version after the restart. Also, suppose the MES client is installed in a battery-running device like a tablet or laptop. Their operation system will only be configured to run updates if the device is connected to the power grid.

As the clients run in more powerful nodes and more complex environments, their update process could be enriched with more proficient methods. The methods are feature flags, dark mode, or A/B testing, which will offer an easy transition to new functionality. The new version will be the same as the previous one upon the switchover, and then new functionalities could be gradually enabled. The end user would increasingly receive new features in this way. In case of a problem, the features could be quickly turned off remotely. Also, new versions of features could be assigned to specific clients to evaluate, following the A/B testing strategy.

5.2. Software Update Approach for Devices with Limited Storage Space

To address this challenge, an additional device of the same type, preferably with a larger storage capacity, is introduced. This backup node is a repository for storing backup versions of the currently running software. In scenarios where the Internet of Things (IoT) layer comprises multiple similar or identical nodes, adding an extra device is not perceived as a drawback but as a justifiable minimal cost.

The same approach applies to Manufacturing Execution System (MES) clients. However, the key distinction lies in the role assigned to the chosen node. In the MES environment, the selected node assumes the mantle of a leading or sentinel client responsible for distributing update packages within its designated group. Utilizing backup nodes at the MES client level is also feasible, especially in cases where stringent IT security protocols prohibit the retention of old software versions due to company policies.

The deployment process commences by transferring the new version (version N) to the backup or sentinel node. Once this operation is completed, the backup node disseminates version N to all devices running the same software. Notably, this approach slightly extends overall downtime, as the target node must first halt the previous version (N – 1), acquire the new version, and subsequently initiate version N. Conversely, no discernible difference in overall downtime occurs when the backup node acts as a sentinel.

An inherent drawback of this approach pertains to increased data traffic requirements. However, this traffic is confined solely to communication between the sentinel or backup node and the clients within its designated group. An additional advantage emerges during potential rollback scenarios. After uploading version N to the backup node, deployment to sensor nodes occurs sequentially. The process begins with the sentinel device (borrowing from the canary deployment concept), where comprehensive validation under production conditions occurs. If the new version proves valid, subsequent nodes receive the update. Conversely, the rollback sequence is limited to the sentinel device if issues arise.

In the second scenario, continuous uptime on the device is not feasible during the update process. Specifically, the currently running version (N-1) must transition to sleep mode and then be removed from the destination device. Subsequently, the new version (version N) is uploaded, configured, and activated using a wake-up command. Until version N is fully operational, the node remains in downtime and temporarily unable to collect or exchange data—an inherent vulnerability that must be managed.

5.3. Software Update in Edge Layer Affecting IoT and MES Nodes

The simple software update at the Edge level would be managed at the other levels by employing message queues. Incoming messages to the Edge level will be handled when it becomes operational again. Messages from the output queues of the Edge level will be processed until Edge components are offline. The connecting systems will raise an alarm if all the items are processed. The same will apply if the incoming buffers become fully loaded.

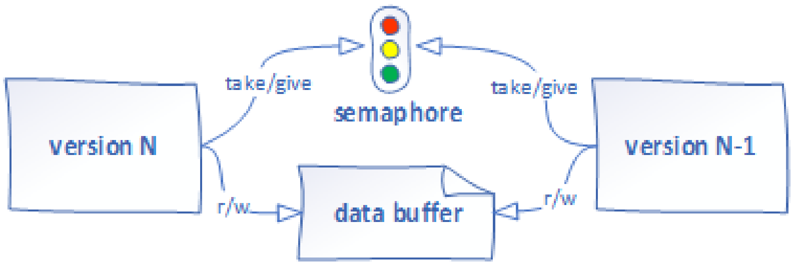


Fig. 9. Software update scheme with message queue [9]

Considering this, it is crucial to define the buffers as wide and long enough to accommodate the amount of data that could be generated during more extended downtimes. In the scenario when the device from the Edge level must remain inactive for a period of deployment and when there are no buffers or message queues implemented, the connected systems will run into an alarm state. Devices at the MES level will raise an alarm, but they will continue executing other actions that are not connected to the Edge level. Some functionalities will be temporarily stopped, but most work could continue.

Devices at the IoT level will not be in such an advantageous position in this case. Without a buffer, devices at the IoT level will get disconnected for the same amount of time as the Edge-level devices. For IoT nodes, this will be a situation of a high alarm state, and they will execute the following course of events:

- Devices in IoT nodes detect disconnection event
- Devices raise the internal alarm
- Start reconnection procedure in predefined time frames

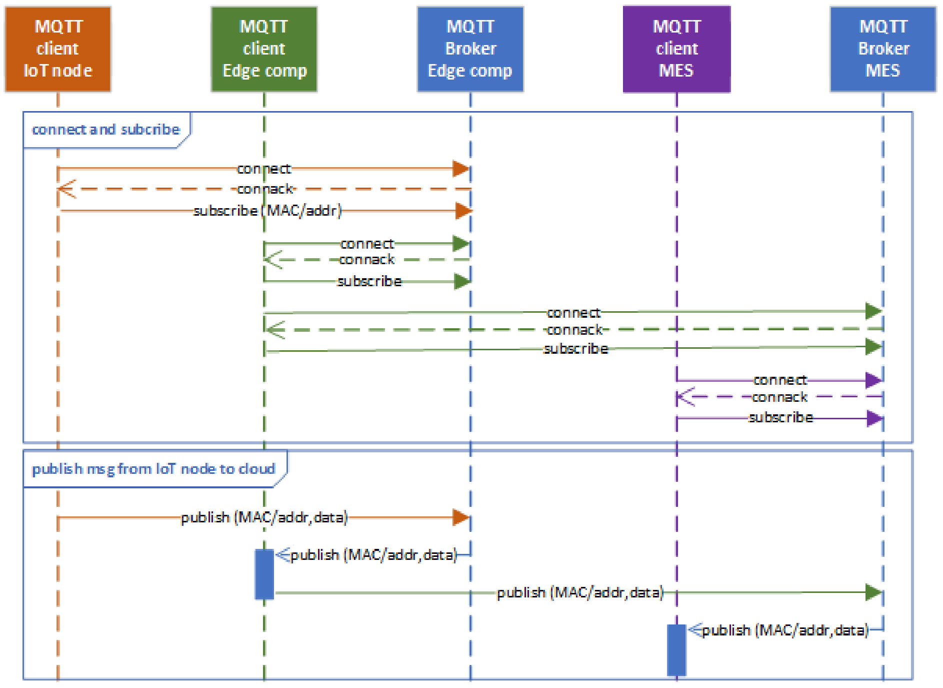


Fig. 10. Reconnection sequence between IoT, Edge, and MES node

Without a buffer enabled, while the Edge level node is not running, IoT nodes will not have a destination where to send processed data. This will cause significant data loss for the complete deployment areas, which could be unacceptable if the process consumes extensive time. This problematic state will last until the Edge layer node starts running again. When a node from the Edge layer restarts and returns online, IoT nodes will connect again and continue exchanging data.

In some cases, IoT nodes will not be able to reconnect due to a change in communication protocol or a hardware error. In these cases, IoT nodes will run a general alarm, and then the Edge node must be moved back to the previous version. When an update is needed in both layers, the update notification signal will stop the general alarm, and then all IoT nodes will be updated individually. The update will be driven from the backup node.

One of the commonly used solutions to reduce the necessity for frequent updates across the levels is the using a buffer between the layers (Fig. 9). In this case, the buffer is implemented as the message queue. In most cases, when the communication protocol changes, only the synchronization buffer will be updated, while all the nodes in the IoT layer will continue to work. In this way, downtime will hit only one layer (in this case, the Edge layer) while the other layers will continue to run without interruptions.

Introducing a message queue solves the previously described issue but at the cost of a bit more complex setup and integration. Fig. 10 This shows the process of integration with the Edge level. The approach is the same for IoT and MES nodes on opposite sides of the Edge node. The Edge nodes establish communication using message queues

(MQTT in the case of the presented system). MQTT brokers and clients are installed at the Edge and the MES level. The IoT node needs only the client.

The connection is initiated from the client on one level to the broker on another. When this communication is established, the broker waits for the client's connection at its level and accepts the subscription request. In this way, MQTT clients in the IoT and Edge levels are connected through the broker at the Edge level. Similarly, MQTT clients from the Edge and MES levels will be connected through the broker at the MES level.

It must be stated that when transferring data using a message queue, data loss could happen during the software update. Message queues usually contain objects of specific types produced on one side and consumed on another. The two most common scenarios are when the connection between the message queue and one of the sides (producer or consumer) cannot be established, while another is when the data queue contains objects of unrecognizable type in the destination. The first situation is handled in a way that stops the producer until the connection is fully re-established. The second situation happens mostly when the version of consumer software is replaced in a way that stops supporting old message formats. In this case, the messages remaining in the queue will be lost. Synchronization through message queues is an essential aspect of the software update, but it goes beyond the scope of this paper.

6. Update Mechanism for MES Nodes

The main shift that could be done at the MES level is integrating the software update mechanism into the solution. MES architecture, which we exploited in our environments, is service-oriented architecture (SOA) based on different technologies. On the server side, multiple services running to achieve necessary functionalities. The current setup is between single service and microservices since the system consists of main execution and multiple supporting services. While the supporting services could be turned on and off independently, the leading execution service must be active to put the system in run mode. In that sense, the update service is one of the services on the server side responsible for server and client updates. Ideally, the update service is configured to run in the independent node. It takes care of the order of the update and data buffers during the update process.

Depending on the requirements, the update service could take care of every single node in the system or equally distribute the updates depending on the node type. The update service takes care of sentinel/backup nodes (if configured) and monitors and switches different feature flags and A/B functionality variants on and off. The approach with the controllable update mechanism, driven from the single node, applies to any ISA95 level. Depending on the technology, implementation could be different, but the concept of maintaining the update process and the configuration from the single point makes the system fully controllable and maintainable. Moving these functionalities from the execution service and its connected microservices to an independent node avoids the well-known problem of the server bottleneck during the update process. In the cases where the execution service itself triggers and controls the update, the network traffic significantly rises during a brief period, which could lead to different synchronization problems.

The additional advantage of implementing such a node is the possibility of connecting it to the digital twin in the cloud. This feature makes updating over the air and synchronization with the digital twin possible. Having such a connection would allow a complete industrial facility to be controlled remotely, and the existing digital twin would always be available for any test and analysis.

Both client and server nodes will use the standard network protocols to operate at the MES and ERP levels. In the lower levels, the accessibility will depend on the implemented technology. Still, with the appropriate network adapters, the update node could achieve control also over the instances in Edge and IoT levels. The update node could also monitor configuration changes in production environments and take adequate action when the change is detected. Depending on the configuration or requirement, it could push the change to a digital twin, raise an alarm for the additional check, or overwrite the configuration.

The additional benefit is the more accessible support for testing and verification before moving the change production environment. As mentioned before, after the solution has been evaluated to a digital twin, test, or staging environment, the deployment for production could be ready significantly faster. The access to configurations already prepared in the digital twin environment allows the update manager to check the destination clients and easily spot if the local changes have been made. In that case, it could stop the deployment and raise the alarm to the technician to decide how to proceed. Alternatively, the update manager could override the configuration in the client machines and force the update.

The update node could also push the update for the server side. At the MES level, the server-side SOA system will also store all the actual and previous versions of the clients, allowing easier recovery and fallback in the case of unsuccessful deployment. In case of the configuration on multiple server instances, the update manager will track the order of the update, using the feature flag system to control the start and stop of all microservices. As mentioned, the leading service on the server side is the execution service required to be active to make the entire system run.

The server side of the update mechanisms is responsible for communicating with clients and other external systems – such as databases, configuration storage, and other external services. It could be configured to retrieve data from multiple sources and prepare the deployment packages according to the status set in the digital twin. As mentioned, its role is also to monitor the validity of the complete system to check if the configurations or client versions may change outside of the deployment process and to raise the alarm in case of misalignments.

Both clients and service exchange ping messages to keep the system communication status. Ping messages could contain distinct parameters and run in different periods. While some are used only to check if there are responses on the other side, others could be used to verify client versions and configurations. At the same time, regular messages that exchange data are used to maintain connectivity. Every message delivery failure could trigger an alarm and run the re-assessment process and eventual network reconfiguration. In some cases, the sentinel clients could take the server role for the group of clients and maintain connectivity in the alarm mode.

6.1. Update Node Routines

The *DeploymentHelper* component handles configuration updates in scenarios where the application reverts to an older version or when a specified time for updating specific clients has elapsed, necessitating updates for the remaining clients. This component is situated on the server side, as all configurations for this application reside on the same machine as the service. Consequently, the service possesses all necessary permissions for file modification and physical addresses where the files are located.

The base class diagram to support client updates is presented in Fig. 11. Instances of class Update Status Info stores the info about the version and application name. The bare minimum of the data should be maintained for every client. They come to the MES or update service as part of ping messages from clients. Combining these pieces of information with the data in the internal cache, the process that keeps track of versions could maintain their activity tables regularly. Activity tables are kept in the update process and periodically synchronized with the digital twin environment. The objects of this class, either persisted in the memory or a dedicated location in the file system, are also used as the contact point for the *DeploymentDispatcher*.

On the single node level, the *DeploymentDispatcher* is the component responsible for the entire update process. It could be configured to ping the server or sentinel client to check for the new version or to wait for the update notification. Once the latest version is discovered, the update process will start and be executed in *UpdateDirector*.

The update thread will run in the background and gather all necessary configurations and binaries from the update node to form the new version of the client. After a new client is formed, it will trigger the rest of the process and perform possible additional steps, such as a backup of the previous version and a blue/green switch. When configured in the sentinel node, this functionality will propagate the installation to other nods in the group. As the ultimate step of the update process, the information about the software version will be pushed back to the update node and the digital twin to ensure the proper version info synchronization.

It is essential to point out that *DeploymentDispatcher* could progress both with complete client updates and partial functionality enabled/disabled. In that way, direct support for feature flags is implemented. The client could come with an updated version of the software, but in case of any problem, the additional features could be disabled. Also, configuration changes could be pushed from the server to ensure the required reconfiguration.

The update manager instance is created when the application is started. It is constantly active and periodically checks for recent updates if configured to run in active mode. During initialization, the update manager checks the application's version and all modules to ensure the up-to-date application signature is ready for comparison with the version on the server.

The update manager listens to the server's ping and notification commands in passive mode. In this scenario, the server notifies the client that the updated version is available, and the client starts the update process. Also, it is usual to configure both modes in the same and dedicate each process to a specific part of the update process. For example, the check for the new client version could be configured in active mode, while the configuration updates could be passive and pushed by the server instead of the client's request.

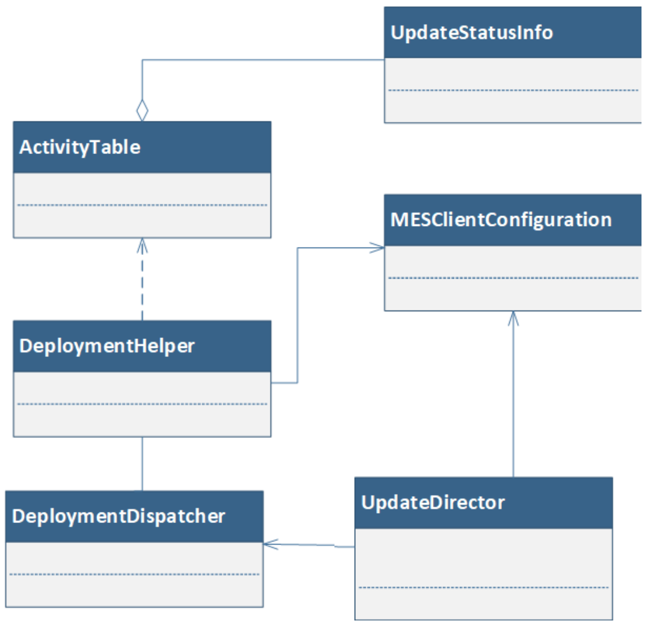


Fig. 11. The relations between main entities in the deployment subsystem

An instance of this class creates an object of the *DeploymentDispatcher* class and immediately invokes its primary function, as shown in Fig. 12. This function manages a specific client's update process and could halt software updates if necessary. It is responsible for initiating the update process as long as the attribute's value that keeps the loop alive remains unchanged.

This method initially attempts to retrieve the file containing the necessary information for updating. If that file does not exist, the method returns a false value, indicating that it failed to obtain the appropriate file. If the file is successfully retrieved, relevant data required for updating is extracted from it. Subsequently, it checks whether beta updates are active. If they are and the specified time for this type of update has elapsed, the UpdatesManifest.xml file is updated. In this file, the active software version is set to the "beta" version, and updates of this type are marked as inactive. Next, it verifies whether the current client version matches the version that should be on our machine (Fig. 12).

New client versions must be downloaded if the current client version is missing or differs from the version in the file while beta updates are inactive. In the case of active beta updates and the client still not being on the beta version, affirmative information is returned to download new files, but only if random access permits. This ensures that not all clients receive the updated value, only those with "luck" (Fig. 12). All clients downloading the updated version exit the function and return a value true. If the random selection does not choose a client, the thread responsible for updating is put to sleep for a predefined number of minutes. Afterward, the thread is again put to sleep for a few seconds, triggering the update check.

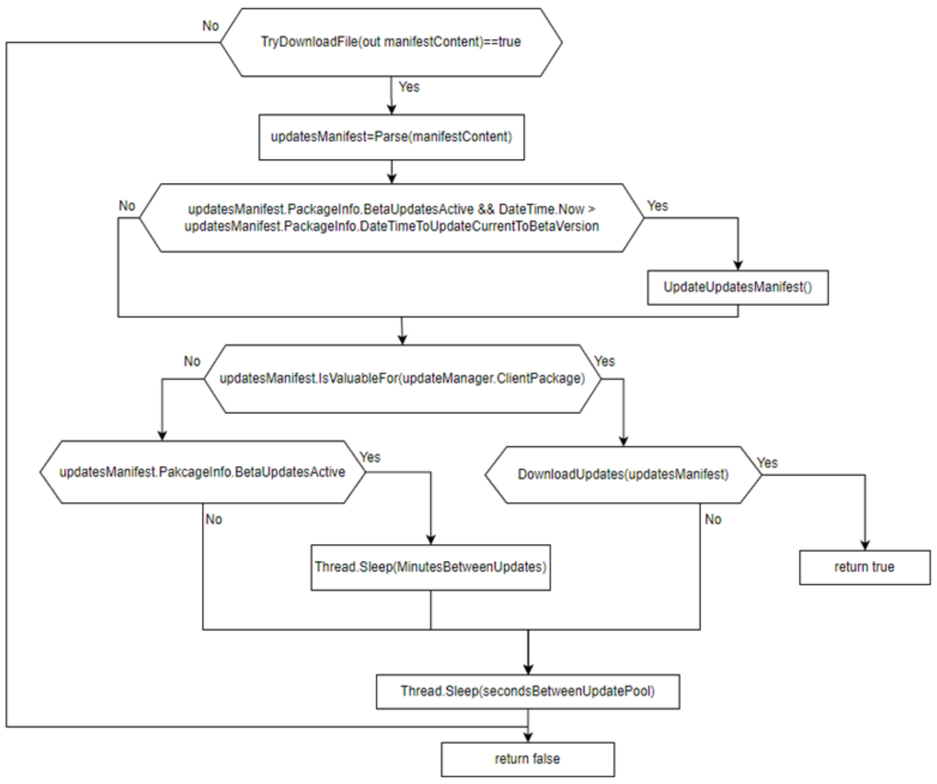

TryDownloadFile(out manifestContent)==true
No
Yes
updatesManifest=Parse(manifestContent)
updatesManifest.PackageInfo.BetaUpdatesActive && DateTime.Now > updatesManifest.PackageInfo.DateTimeToUpdateCurrentToBetaVersion
No
Yes
UpdateUpdatesManifest()
updatesManifest.IsValuableFor(updateManager.ClientPackage)
No
Yes
updatesManifest.PakcageInfo.BetaUpdatesActive
Yes
No
DownloadUpdates(updatesManifest)
Yes
No
Thread.Sleep(MinutesBetweenUpdates)
return true
Thread.Sleep(secondsBetweenUpdatePool)
return false

Fig. 12. The sequence of choosing and verifying the correct software version

The *DownloadUpdates* method retrieves updates from the corresponding file (the file path is specified in the update specification). If beta updates are active, it fetches the file named in *BetaFilePath*; otherwise, it retrieves the file named in *FilePath*. *BetaFilePath* is used when the A/B deployment must be supported, while for regular deployments, the filed *FilePath* directs to the update location. This approach also solves the issue of network connection interruptions to the new client, as the update is not applied until it is fully downloaded locally. Finally, the application that launches the latest client version is restarted.

7. Results and Discussion

This research came out of the project that resulted in the development of a complex industrial monitoring system aimed at all ISA95 levels – from IoT nodes through Edge and MES to ERP level. During the project, for more than 15 years, our team was focused on different aspects of development and implementation, starting from the improvements of CAD/CAM databases [31] through all different implementations at all levels, up to development for the software update system integrated with the cloud [9] [10].

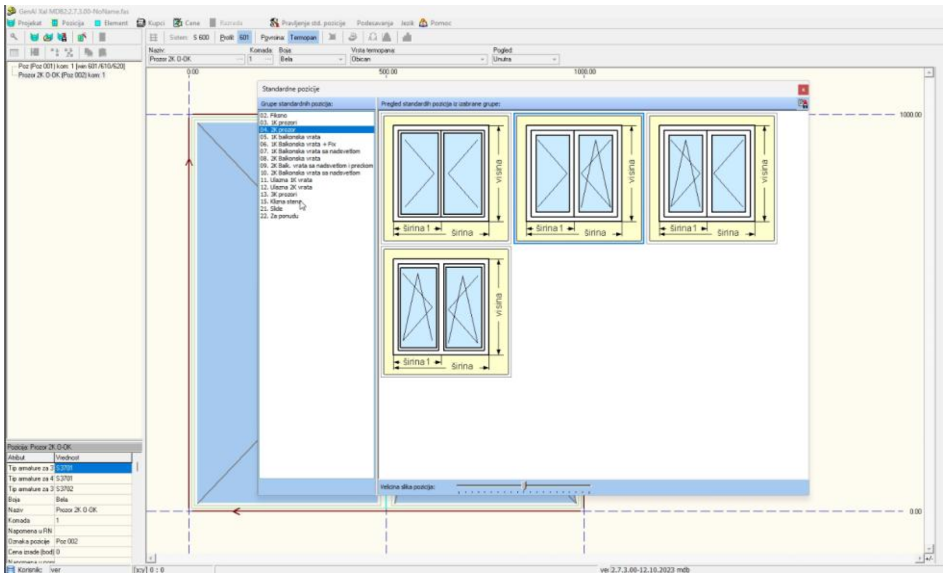

Standardne pozicije
Grupe standardnih pozicija:
Pregled standardnih pozicija iz izabrane grupe:
visina
širina 1
širina
Veličina slike pozicija:

Fig. 13. View on the ERP client - production order definition

The tests are conducted in a digital environment that resembles the industrial façade carpentry facility. Section 4 gives all the necessary details in the composition of the test environment. Such production is interesting since it combines different production types – from serial production to one-of-a-kind configured products [32]. At the same time, such a facility combines processes based on various physical and chemical procedures in material treatment, thus requiring all kinds of digital interaction, starting from thermal sensors and actuators through intelligent industrial machines integrated with MES clients up to ERP software enhanced with different CAD and planning tools (Fig. 13) [33].

Having experience with diverse types of software developed on different ISA95 levels, we identified the common problems in software updates and tend to generalize the update architecture, node structure, and processes. The results were preliminarily evaluated at the IoT and ERP levels because they have limited effects on the rest of the system, being connected only to the neighboring level. Following the results and recommendations from the previous work, we decided to expand the update system to the most challenging MES level (Fig. 14).

7.1. Guidelines for Combining Different Deployment Strategies

Our research was led by the request to reduce the potential downtime during the software update in a challenging environment such as the industrial facility. The actual criticality of this request is not equal from level to level, but the customer requirement tends to go to 0 downtime regardless of the software system. To reach this goal, we decided to replace the standard deployment (stop-copy-run) with a combined strategy that should employ the benefits from different deployment processes. Looking at the single node, we aimed for the blue/green deployment as the base concept.

This concept could be enriched then with feature flags, dark mode, and A/B testing deployments to fine-tune the update process and to release new functionalities in the controllable environment. At the level of the node networks, the concepts of canary deployment were applied to the development of backup and sentinel nodes, which function as the group leads and will receive the first update and then push forward deployment into the subsequent nodes in its group. Combining these three well-known approaches in the proposed way, we tried to benefit from all the positive aspects we could get:

- Blue-green deployment gives the possibility for a fast version switch.
- Dark mode and feature flags allow simple enabling or turning off single functionalities.
- A/B testing allows running several feature variants to let the customer decide which to accept.
- Canary deployment allows prompt identification of deployment errors.
- A synchronization buffer allows us to keep one layer insulated and operative while the connected layers are in downtime or performing an update.

The proposed methodology is initially subjected to rigorous testing at the IoT level. This choice stems from the formidable constraints encountered in this stratum, encompassing software resources, network bandwidth, and energy consumption limitations. Additionally, deploying IoT systems in critical and hazardous environments underscores the need to minimize direct human intervention and avoid installing supplementary infrastructural components, such as power or network cables.

Complicating matters further, physical access to IoT nodes remains a challenging endeavor. This challenge arises not solely from technological considerations but also from mechanical and security protocols. Removing various mechanical elements in certain instances becomes necessary to reach IoT devices physically. Moreover, these devices often operate in environments hazardous to human safety, necessitating stringent procedures for device access.

Previously, a conventional update approach, or recreate deployment, was employed, wherein the software component was replaced either entirely or partially (via a stop-copy-start process). However, this standard update method posed several issues, which can be briefly summarized as follows:

- The downtime was always present. If the software component is in the updating process, the software device cannot be used.
- In case of an erroneous update, software should be restored to its previous version, which would lead to further downtime.
- The restore process sometimes drains the battery, requiring the personnel member to go to the hazardous area.
- Connected layers generally could not continue to work since they were flooded with alarm signals.

Our results with the proposed combined deployment approach proved our expectations and varied between different software layers and scenarios. Applying the proposed strategy reduced the overall downtime and number of unnecessary rollbacks. This was achieved by the cost of implementing the backup node, the implementation of the buffer level, and a slight increase in data traffic. Table 3 shows the behavior of the network of 100 IoT nodes analyzed in a test environment.

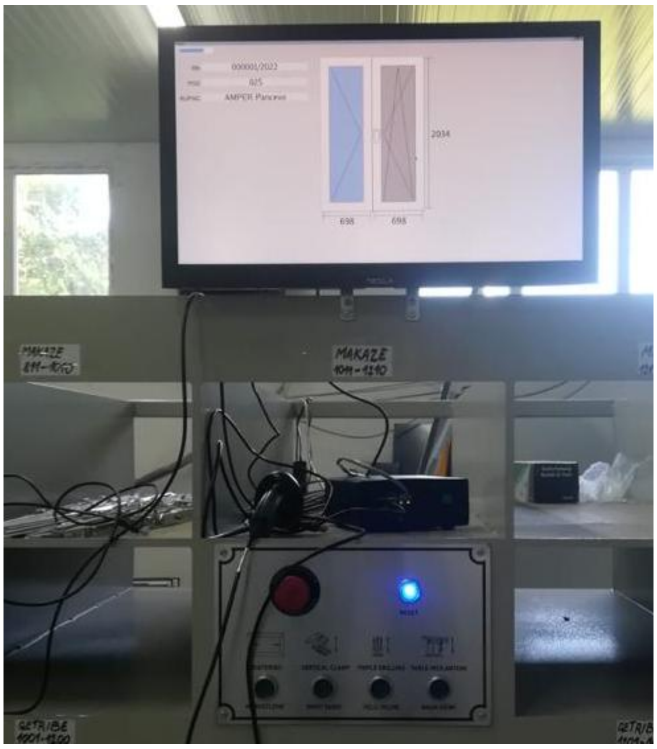


Fig. 14. MES client set up in a factory environment - connected to cutting machine and the signals that bring measurement values

Having the configuration with one leading node, the total number of updates coming from the update node or the cloud to the IoT network will be reduced from the total number of nodes (NN in further text) to one. The updated version will come from the outside system to update the node, which will guide the update for the rest of the IoT nodes. In this way, the bottleneck in communication between the IoT level and the rest of the system will be reduced or eventually avoided. This way, the number of security checks will be reduced to only one. In a scenario where every node gets an update outside the network, a security check will be performed every time due to standard security policies.

The proposed hybrid approach will require more space. If the clients can support blue/green deployment, they will need twice as much space as in the case of recreate deployment. One additional slot for the distributed version should be added to the space required. The sentinel client will use the distribution/sentinel/backup node to download the updated version and then forward the update.

Table 3. The effects of the proposed deployment strategy on the IoT level containing 100 IoT nodes connected to a single Edge node (TD – time to shut down the software in the node, TU – time to start the software in the node, TS – time switch between the versions, IS – software instance size per node, NN – number of nodes). Combined from [9] and [10]

Measurement	With recreate deployment	With hybrid strategy
Number of software uploads to IoT level – successful deployment	NN	1 (only to the leading node)
Number of internal uploads – successful deployment	0	NN
Number of software uploads - unsuccessful deployment	Average 8% of NN	1 to the backup node
Security check on upload	NN	1 (only to backup node)
Number of internal software uploads – unsuccessful deployment	0	1
Rollbacks with unsuccessful deployments	8% of NN	1 + 1
Downtime per node	TD + TU (in seconds)	TS (in milliseconds)
Used space for software per node (with blue-green approach)	1 x IS	2 x IS
Used space for software with buffer node	NN x IS	NN x IS + IS
Update distribution	Manual or with a task scheduler	Optimized by backup node or pushed from the cloud
Downtime when connected layer update	If the update is running	Until the buffer has data

The concept proposed for IoT nodes in [10] further evolved and applied to the ERP nodes [10]. With further customization, it is successfully applied to the MES level. The expected effect is presented in Table 4. Both ERP and MES clients share similarities in size and software architecture. Both have more extensive software instances than those in the IoT and Edge levels. Due to the software's mentioned size, update distribution could cause problems comparable to those from the IoT level, primarily if the update is run from the same node where the server is running. In that case, the single node should run NN uploads, which could take significant network resources.

To address this challenge, a strategic division of client nodes into N1 groups by AG clients is proposed (Fig. 15). This approach draws inspiration from the canary deployment methodology, wherein a dedicated group of clients serves as the initial testing cohort. During the first iteration, updates are dispatched to sentinel nodes, responsible for essential testing. Subsequently, these sentinel nodes propagate the verified updates to the nodes within their respective groups. In the event of an error detected at the sentinel level, a rollback ensues, ensuring that most clients remain shielded from erroneous software versions. This approach undergoes slight adaptation when applied to the MES layer. The rationale behind this modification lies in the inherent diversity of MES clients. Unlike ERP clients, which typically exhibit uniform features, MES clients cater to distinct operational stations, each potentially possessing a significantly separate set of functionalities. In the MES environment, an initial client group is selected for deployment. The updated version is relayed to its sentinel node, where thorough verification occurs. Upon successful verification, the updated version cascades to the remaining group members. Subsequently, the verified functionality extends to other sentinel nodes.

Table 4. The estimated effects of the proposed deployment strategy in MES and ERP level (TD – time to shut down the software in the node, TU – time to start the software in the node, TS – time switch between the versions, TF – time needed to activate feature flags and A/B features, IS – software instance size per node, BS – buffer size, NN – total number of nodes, N1 – number of level 1 nodes (sentinel/backup nodes), G – number of level 2 groups, AG – average number of level 2 nodes per group AG = (NN – N1)/G)

Measurement	Recreate deployment	Hybrid deployment ERP level	Hybrid deployment MES level
Number of software uploads to level 1 nodes – successful deployment	NN	N1	1 + (N1 – 1)
Number of software up-loads to level 2 nodes (average per group, successful deployments)	0	AG	AG
Number of software uploads to level 1 (rollback needed)	NN	Up to N1	1
Number of software uploads to level 2 (rollback needed)	0	0	AG
Security check on upload	NN	N1	1 Only in the update node
Downtime per node	TD + TU	TS	TS + TF
Total space used	NN x IS	NN x (2 x IS + BS) + IS	NN x (2 x IS + BS)
Update distribution	Manual or with a task scheduler	Optimized by backup node	Over the air
Downtime when connected layer update	If the update is running	Until the buffer has data	0 – ERP Until the buffer has data – Edge / IoT

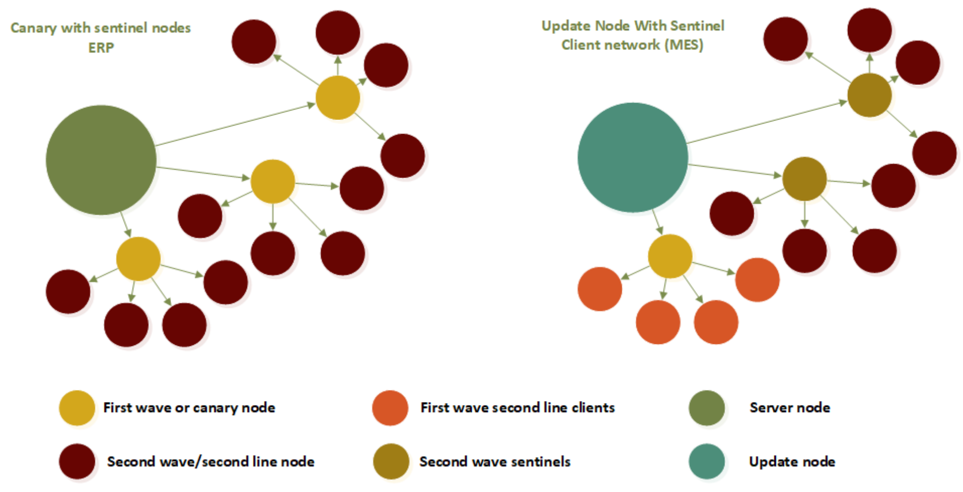

Canary with sentinel nodes ERP
Update Node With Sentinel Client network (MES)
First wave or canary node
First wave second line clients
Server node
Second wave/second line node
Second wave sentinels
Update node

Fig. 15. Differences in deployment approach for ERP (left, as presented in [10]) and MES clients (right)

Table 5. Effects of different client deployment approaches to MES and ERP level – 3 groups of 10 clients (STD – standard approach, WoD – Wave of Distribution, RD – Recreate Deployment, HD – Hybrid Deployment, CwS – Canary with Sentinel, GwS – Groups with Sentinel)

Measurement	ERP RD	ERP HD CwS	MES RD	MES HD GwS
Number of update packages sent from the server to clients (1st WoD)	30	3	30	1 + 2
Amount of data sent from the server to clients (in GB, 1st WoD)	1.35	0.14	0.75	0.03 + 0.07
Network traffic peak (in %, server outbound, 1st WoD)	100	18.65	78.40	5.67
Distribution group size (2nd WoD)	-	10	-	10
Distribution time per group of clients (seconds, 1st WoD)	64.28	7.55	41.19	2.77 + 6.01
Distribution time per group of clients (seconds, 2nd WoD)	-	17.08		12.55
Single client switchover/update time (seconds)	32.28	4.58	25.19 (MES only) 31.22 (complete)	2.41 (MES only) 8.67 (complete)
Single client switchover/restart time when rollback is needed (seconds)	34.10	6.78	26.49 (MES only) 33.53 (complete)	4.33 (MES only) 9.02 (complete)

While this approach does not directly reduce total network traffic, it effectively distributes the load across update and sentinel nodes, mitigating network traffic hotspots. Anticipated downtime per node may be slightly higher for MES clients due to the activation of feature flags and A/B functionalities. Additionally, depending on configuration, MES clients may require time to establish connections with signal sources from distinct levels. Notably, integrating the update mechanism with the Cloud level and the digital twin introduces the prospect of fully controllable over-the-air deployment, potentially paving the way for a transition to software-as-a-service for specific system elements.

We compared the update behavior for the array of 30 ERP and 30 MES clients running in the test environment to evaluate predicted values. They have been split into three groups of ten clients for the simulation. The findings, presented in Table 5, align with the estimation from Table 4. Due to their smaller size, MES clients create less network traffic than ERP clients. The amount of required space and network peaks are lower for the MES network.

7.2. Advantages and Drawbacks

The advantage of the approach shown in this work is that if it is applied to MES nodes, it results in faster recovery if the deployment error is noticed, compared to the one presented in [9] and [10]. Usually, it is enough to do the rollback only in one sentinel node. The next advantage is the possibility of running multiple versions of some functionality and quickly switching them on or off. Ultimately, integrating with cloud services and establishing a complete digital twin helps detect errors and change. The environment we used for the test is a demo digital twin for beta testing.

It is essential to note that two separate times must be measured when the MES client is started or when the switchover is handled. The most critical moment is when the client is in running mode and connects to the MES service, allowing it to perform standard MES functionality – operation execution, labor logging, etc. Next is the moment when the client is connected to other data sources. In our example, clients are connected to an OPC (object for process control) server that acts as a system that collects measurements from the sensors. Generally, these data sources could be different depending on the area of the industrial facility where the client is running.

The software update challenges discussed in this study constitute only a portion of the broader complexity. For over fifteen years, we have continuously relied on systems developed by our research group, honed through rigorous coordination, and field-tested in partner industrial facilities. The software update process encompasses several critical dimensions, including compatibility concerns, system stability, data migration intricacies, and the imperative of user adoption. Addressing compatibility issues necessitates comprehensive testing across diverse system configurations before deployment.

To this end, we advocate for establishing a dedicated test environment within our domain or creating a digital twin in the cloud. For instance, transitioning to a different platform version for Windows application development may introduce incompatibilities with OPC servers. Similarly, upgrading the database server to a newer version could disrupt continuous connectivity between MES or ERP systems until the connection driver is updated. Altering the data structure of messages stored in message queues poses the risk of data loss for existing records, rendering them unreadable by the current system.

User adoption hinges on effective communication and targeted training to elucidate the benefits of updates and familiarize users with new features. Soliciting feedback from users both before and following updates facilitates the identification and resolution of any emerging issues. The strategic inclusion of A/B deployment techniques further enhances this process.

The typical application of the proposed software update mechanisms is limited to some point. This means that the suggested set of updates could not be directly used for software not developed in the line of the examined software development and deployment approaches. For example, if the software has no properly exposed extension and configuration classes, there will not be the possibility to use feature flags or A/B approaches. On the other hand, blue/green and canary deployments could be implemented through a committed team supported with the necessary hardware and acquiring specific deployment routines. A deeper implementation of the proposed deployment solution would require additional pieces of software and/or additional adaptation in the target software.

During the development process, not all pieces of software were designed suitably and flexibly for such update mechanisms. Initially, the MES software was developed with fixed configuration files in which content was loaded on system startup, and the update was not possible while the software was running. This was primarily related to the server side. Any configuration change used to lead to service restart, which eventually results in execution disruption. For this reason, the blue/green deployment was the first included in the setup. It guaranteed reduced downtime and faster system operational availability. On the other hand, the software adaptation for MES clients came a bit later since it only needed to restart local clients in the operator's place, which had a limited impact. The next set of updates was the approach that could trigger configuration refresh through a database or file reload. With this approach, feature flags and later approaches became fully supported, and the software was ready to become a part of the complex deployment system, significantly reducing downtime when redeployed.

Mitigating system disruptions involves judiciously scheduling updates during off-peak hours and transparently communicating potential downtime to users. Meanwhile, prudent planning and rigorous testing of data migration procedures minimize complications arising from data transfer.

In summary, a carefully orchestrated update process, underpinned by thoroughly vetted software versions and executed at the opportune moment, constitutes the linchpin of a successful upgrade.

8. Conclusion

Having more than a decade and a half of experience with industrial systems, our research team went through different projects involving software development at all ISA95 levels. The challenges in development vary across the levels due to user requirements, technical complexity, and performance expectations. All these software instances must work in accordance and be a reliable element of the industrial facility. The common challenge for all the pieces of software is the system update. Usually, the system on one level consists of the server and several dozen or hundreds of clients. When it comes to the update, it should be done as fast as possible and with lower resource consumption without creating bottlenecks in the facility.

The research findings significantly advance the formulation of deployment strategies for intricate, layered industrial software systems. When deploying software updates, several common challenges arise, including downtime, increased network traffic, and storage space utilization. At lower levels, energy consumption during the deployment process also warrants consideration.

We introduce additional backup nodes into the system to address the limited storage space issue. Although these backup nodes exhibit a slightly larger volume than regular IoT nodes, this tradeoff is deemed acceptable given the achieved outcomes. Notably, total downtime has been dramatically reduced—from seconds to milliseconds—representing a reduction of less than one percent of the initial duration.

The approach used in IoT nodes [9] was successfully applied to ERP [10] and MES levels by improving the defined hybrid deployment mode. The findings align with those observed for IoT nodes, emphasizing the potential incorporation of novel features and deployment strategies. This adaptability makes the deployment process for ERP and MES clients more user-friendly, fostering higher user acceptance rates.

We devised a hybrid strategy combining blue-green, canary, and dark mode elements with feature flags, A/B testing, and enhanced standard deployments. This strategy is bolstered by an inter-layer buffer and the inclusion of specific nodes—the update node on the server side and backup and sentinel nodes on the client side. By implementing this approach, we effectively curtailed overall downtime, reducing the duration required for system restart to a period proximate to the switchover. Remarkably, this reduction translates to less than 10% of the time typically consumed by classic deployment methods. The most noticeable improvement is in the case of erroneous deployment when the error could be tracked down and stopped in the first sentinel node.

With the backup/sentinel node active, we reduced the number of software uploads in case of an erroneous update to the time needed for two switchovers of the single node. If chosen correctly, the initial sentinel node will provide an adequate test environment for error detection. Unlike the ERP clients, where the approach was to release the update to all sentinel nodes, with MES clients, the strategy was to send the update to a single sentinel, and then it would take care of its group. In the worst case, the targeted group needs to be reverted, but this will be done inside the group without the need for interaction with the server or the update node.

The changes in the deployment process applied to MES nodes are driven mainly by the Industry 4.0 paradigm and the requirements that came with it. MES and Industry 4.0 are transforming manufacturing practices by digitizing and making processes intelligent, enabling organizations to cater to individual customer requirements and achieve operational excellence. In short, MES and Industry 4.0 are revolutionizing manufacturing by integrating advanced technologies and data-driven systems to create a more interconnected and efficient production environment.

Enhancing the efficiency of the software update process stands as a pivotal element within an optimized production environment. The overarching objective is facilitating software updates beyond scheduled maintenance windows. Leveraging the proposed hybrid deployment method, seamless layer-wide updates become feasible, particularly when interactions with other levels remain unchanged. Notably, this approach significantly truncates downtime—from hours and minutes to mere seconds and milliseconds. Furthermore, our future trajectory involves extending our efforts to the Edge level. This strategic expansion aims to devise solutions that mitigate the impact of buffering and inter-level communication system modifications more effectively.

Acknowledgment. CERCIRAS supported this work – COST Action CA19135, funded by COST. The Ministry of Science, Technological Development, and Innovation of the Republic of Serbia supported this work [grant number 451-03-65/2024-03/200102].

References

1. ISA95, Enterprise-Control System integration- ISA (no date) isa.org. [Online]t: https://www.isa.org/standards-and-publications/isa-standards/isa-standards-committees/isa95 (current October 2024).
2. Shu, Z., Wan, J., Zhang, D., Li, D.: Cloud-integrated cyber-physical systems for complex industrial applications. Mobile Networks and Applications 21.5 865-878. (2016):
3. Kondratenko, Y., Kozlov, O., Korobko, O., Topalov, A.: Complex industrial systems automation based on the internet of things implementation. In Information and communication technologies in education, research, and industrial applications (pp. 164–187). Springer International Publishing. https://doi.org/10.1007/978-3-319-76168-8_8. (2018)

4. Sha, K., Errabelly, R., Wei, W., Yang, T. A., Wang, Z.: EdgeSec: Design of an edge layer security service to enhance iot security. In 2017 IEEE 1st international conference on fog and edge computing (ICFEC). IEEE. https://doi.org/10.1109/icfec.2017.7 (2017)
5. Li, H., Ota, K., Dong, M.: Learning iot in edge: Deep learning for the internet of things with edge computing. IEEE Network, 32(1), 96–101. https://doi.org/10.1109/mnet.2018.1700202. (2018)
6. Sajid, A., Abbas, H., Saleem, K.: Cloud-Assisted iot-based SCADA systems security: A review of the state of the art and future challenges. IEEE Access, 4, 1375–1384. https://doi.org/10.1109/access.2016.2549047. (2016).
7. Urbina Coronado, P. D., Lynn, R., Louhichi, W., Parto, M., Wescoat, E., Kurfess, T.: Part data integration in the Shop Floor Digital Twin: Mobile and cloud technologies to enable a manufacturing execution system. Journal of Manufacturing Systems, 48, 25–33. https://doi.org/10.1016/j.jmsy.2018.02.002. (2018)
8. Chofreh, A. G., Goni, F. A., Klemeš, J. J., Malik, M. N., Khan, H. H.: Development of guidelines for the implementation of sustainable enterprise resource planning systems. Journal of Cleaner Production, 244, 118655. https://doi.org/10.1016/j.jclepro.2019.118655. (2020)
9. Rajković, P., Aleksić, D., Janković, D., Milenković, A., Đorđević, A.: Resource Awareness in Complex Industrial Systems–A Strategy for Software Updates. In Proceedings of the First Workshop on Connecting Education and Research Communities for an Innovative Resource Aware Society (CERCIRAS), Novi Sad, Serbia (Vol. 2). https://ceur-ws.org/Vol-3145/paper10.pdf. (2021)
10. Rajković, P., Aleksić, D., Djordjević, A., Janković, D.: Hybrid software deployment strategy for complex industrial systems. Electronics, 11(14), 2186. https://doi.org/10.3390/electronics11142186. (2022)
11. Cozzani, V., Antonioni, G., Landucci, G., Tugnoli, A., Bonvicini, S., Spadoni, G.: Quantitative assessment of domino and NaTech scenarios in complex industrial areas. Journal of Loss Prevention in the Process Industries, 28, 10–22. https://doi.org/10.1016/j.jlp.2013.07.009. (2014)
12. Chen, Y., Chen, J., Gao, Y., Chen, D., Tang, Y.: Research on software failure analysis and quality management model. In 2018 IEEE international conference on software quality, reliability and security companion (QRS-C). IEEE. https://doi.org/10.1109/qrs-c.2018.00030. (2018)
13. Usman, M., Felderer, M., Unterkalmsteiner, M., Klotins, E., Mendez, D., Alégroth, E.: Compliance requirements in large-scale software development: An industrial case study. In Product-Focused software process improvement (pp. 385–401). Springer International Publishing. https://doi.org/10.1007/978-3-030-64148-1_24. (2020)
14. Kalunga, J., Tembo, S., Phiri, J.: Industrial internet of things common concepts, prospects and software requirements. International Journal of Internet of Things, 9(1), 1-11. (2020)
15. Chen, C., Reniers, G., Khakzad, N.: A thorough classification and discussion of approaches for modeling and managing domino effects in the process industries. Safety Science, 125, 104618. https://doi.org/10.1016/j.ssci.2020.104618. (2020)
16. Ren, Z., Chen, C., Zhang, L.: Security Protection under the Environment of WiFi. In 2017 international conference advanced engineering and technology research (AETR 2017). Atlantis Press. https://doi.org/10.2991/aetr-17.2018.11. (2018)
17. Kim, D.-Y., Kim, S., Park, J. H.: Remote software update in trusted connection of long range iot networking integrated with mobile edge cloud. IEEE Access, 6, 66831–66840. https://doi.org/10.1109/access.2017.2774239. (2018)
18. Asokan, N., Nyman, T., Rattanavipanon, N., Sadeghi, A.-R., Tsudik, G.: ASSURED: Architecture for secure software update of realistic embedded devices. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 37(11), 2290–2300. https://doi.org/10.1109/tcad.2018.2858422. (2018)

19. Mugarza, I., Parra, J., Jacob, E.: Cetratus: A framework for zero downtime secure software updates in safety-critical systems. Software: Practice and Experience, 50(8), 1399–1424. https://doi.org/10.1002/spe.2820. (2020)
20. Stevic, S., Lazic, V., Bjelica, M. Z., Lukic, N.: IoT-based software update proposal for next generation automotive middleware stacks. In 2018 IEEE 8th international conference on consumer electronics - Berlin. IEEE. https://doi.org/10.1109/icce-berlin.2018.8576241. (2018)
21. Mirhosseini, S., Parnin, C: Can automated pull requests encourage software developers to upgrade out-of-date dependencies? In 2017 32nd IEEE/ACM international conference on automated software engineering (ASE). IEEE. https://doi.org/10.1109/ase.2017.8115621. (2017)
22. Fowler, M.: Blue-green deployment, March (2010). [Online]. http://martinfowler.com/bliki/BlueGreenDeployment.html (current October 2024).
23. Tarvo, A., Sweeney, P. F., Mitchell, N., Rajan, V. T., Arnold, M., Baldini, I.: CanaryAdvisor: A statistical-based tool for canary testing (demo). In ISSTA '15: International symposium on software testing and analysis. ACM. https://doi.org/10.1145/2771783.2784770. (2015)
24. Killi, B. P. R., Rao, S. V.: Towards improving resilience of controller placement with minimum backup capacity in software defined networks. Computer Networks, 149, 102–114. https://doi.org/10.1016/j.comnet.2018.11.027. (2019)
25. Vincent, L.: Marketing strategies for commercialization of new technologies. In Advances in the study of entrepreneurship, innovation & economic growth (pp. 257–287). Emerald Group Publishing Limited. https://doi.org/10.1108/s1048-473620160000026009, (2016)
26. Pleshko, L. P., Heiens, R. A.: The contemporary product-market strategy grid and the link to market orientation and profitability. Journal of Targeting, Measurement and Analysis for Marketing, 16(2), 108–114. https://doi.org/10.1057/jt.2008.2. (2008)
27. Buzachis, A., Galletta, A., Celesti, A., Carnevale, L., Villari, M.: Towards osmotic computing: A blue-green strategy for the fast re-deployment of microservices. In 2019 IEEE symposium on computers and communications (ISCC). IEEE. https://doi.org/10.1109/iscc47284.2019.8969621. (2019)
28. Mampage, A., Karunasekera, S., Buyya, R.: A holistic view on resource management in serverless computing environments: Taxonomy and future directions. ACM Computing Surveys. https://doi.org/10.1145/3510412. (2022)
29. Chien, C.: What is rapid application development (RAD)? (2020). [Online]. https://codebots.com/app-development/what-is-rapid-application-development-rad (current October 2024).
30. Munikanth: Kubernetes Deployment Strategies, Medium. (2023). [Online]. https://medium.com/@munikanthtech/kubernetes-deployment-strategies-fc1557d21e8f (current October 2024).
31. Aleksić, D., Janković, D.: The use of scripts in a CAD/CAM database. In The X International Conference on Information, Communication and Energy Systems and Technologies (ICEST 2009), June (pp. 25-27). (2009)
32. Aleksic, D., Jankovic, D., Rajkovic, P.: Product configurators in SME one-of-a-kind production with the dominant variation of the topology in a hybrid manufacturing cloud. The International Journal of Advanced Manufacturing Technology, 92(5-8), 2145–2167. https://doi.org/10.1007/s00170-017-0286-1. (2017)
33. Aleksić, D., Janković, D. Stoimenov, L.: A case study on the object-oriented framework for modeling product families with the dominant topology variation in the one-of-a-kind production. Int J Adv Manuf Technol 59, 397–412, https://doi.org/10.1007/s00170-011-3466-4. (2012)
34. Rajković, P., Aleksić, D., Janković, D.: The Implementation of Battery Charging Strategy for IoT Nodes. In: Zeinalipour, D., et al. Euro-Par 2023: Parallel Processing Workshops. Euro-

Par 2023. Lecture Notes in Computer Science, vol 14352. Springer, Cham. https://doi.org/10.1007/978-3-031-48803-0_4. (2024)

Petar Rajković is an Assistant Professor at the University of Niš, Faculty of Electronic Engineering. He obtained his Ph.D. in software engineering from the same university and teaches various courses at all levels of study. He is focused on model-driven development and information system research, with practical experience in developing innovative software solutions for industrial automation and public health.

Dejan Aleksic is an Associate Professor at the Faculty of Sciences and Mathematics of the University of Nis, Serbia. He obtained his Ph.D. in software engineering from the Faculty of Electrical Engineering at the same university. His research interests include Product configuration, Mass Customization, One-of-a-kind production, and Industrial IoT.

Dragan S. Janković received a B.Sc., M.Sc., and a Ph.D. in computer science from the Faculty of Electronic Engineering, University of Niš, Serbia, in 1991, 1995, and 2001, respectively. Currently, he works as a full professor at the Department of Computer Science, Faculty of Electronic Engineering, and head of the Laboratory for Medical Informatics. His research interests include logic design, software development, algorithms, medical informatics, artificial intelligence in medicine, and blockchain technology. He was a participant or project leader in more than 30 research and development projects. He published over 350 scientific papers and 10 technical solutions.

Aleksandar Milenković is an Assistant Professor in the Faculty of Electronic Engineering at the University of Nis. He has over ten years of experience in modelling, developing, and implementing medical information systems. He holds a Doctor of Science degree in computer science. His current research interests include medical informatics, medical information systems, and machine learning in medicine.

Andjelija Djordjevic is a Teaching Assistant at the Department of Computer Science, Faculty of Electronic Engineering, University of Nis. She is a PhD student at the Faculty of Electronic Engineering since 2020. She obtained her master's and bachelor's degrees at the same faculty in 2020 and 2019, respectively. Her research interests include software engineering, algorithm design and analysis, and medical informatics. She works on a project dedicated to the development of manufacturing execution systems and is a member of the Laboratory for Medical Informatics at the Faculty of Electronic Engineering.

Received: December 20, 2019; Accepted: May 02, 2020

Computer Science and Information Systems 22(2):437–457 https://doi.org/10.2298/CSIS240725025T

Comprehensive Approach to the Design of Information Systems and Optimization of Technical Solutions according to Many Criteria ⋆

Mikhail Tatur[1], Natalia Novoselova[2] and Marina Lukashevich[3]

[1] Belarusian State University of Informatics and Radioelectronics
Pietrusia Brouki, 6, Minsk 220013 Belarus
tatur@bsuir.by
[2] United Institute of Informatics Problems
Surganova 6, Minsk, 220012, Belarus
novos65@gmail.com
[3] Belarusian State University
Nezavisimosti Avenue 4, Minsk, 220030, Belarus
LukashevichMM@bsu.by

Abstract. One of the problems of the modern information society is the development of effective complex multiprocessor information systems, taking into account the rational use of system resources. The problem of finding the optimal variant of a multiprocessor system is presented as a problem of multicriteria optimization and makes it possible to search for a trade-off between several alternatives. The paper describes several approaches to the comparative assessment of multiprocessor systems, including the search for the non-dominated solutions; narrowing down the Pareto space, using the additional expert information; converting the problem to single-criteria optimization with convolution of criteria; searching for the optimal solution that is closest to the reference point. In the paper the authors propose a ranking method to evaluate technical solutions. The method is based on ranking individual alternatives for each optimization criterion separately, followed by aggregation of ordered ranked lists. The advantage of using the ranking methods is to obtain a complete rating of technical solutions based on their effectiveness, assessed by several criteria. The paper has an educational character and considers the problem of finding a trade-off between system parameters when looking for technical solutions. The practical results of applying different approaches are demonstrated using a simple example.

Keywords: multiprocessor information systems, multicriteria optimization, genetic algorithm, non-dominated solutions, alternatives ranking.

1. Introduction

One of the problems of the modern information society is the development of effective complex multiprocessor information systems, taking into account the rational use of system resources [18]. Examples of such complex systems are global search systems, new

⋆ This manuscript is an extended version of a paper published in proceedings of CERCIRAS WS01: 1st Workshop on Connecting Education and Research Communities for an Innovative Resource Aware Society.

generation information systems “smart home”, “smart city”, “smart government”, cyber-physical systems of autonomous transport, robotic enterprises, etc. Different types of system resources can be considered, including technical resources - functional complexity, algorithmic complexity, hardware complexity, computing performance, mass-dimensional characteristics, energy consumption, etc.; operational resources - reliability, safety, acceptable service life, etc.; economic resources – cost of the system, cost of maintenance, cost of development support and modifications, etc. It is obvious that most of the resources available for optimization are closely interconnected and their rational distribution during design is only possible using a comprehensive approach.

A comprehensive approach and the search for a trade off between the key parameters of the system for new technical solutions include the following subtasks:

– determining the scope of possible technical solutions when using a given method (or architecture, or concept) of implementation;
– scientific justification of objective limitations in the design of complex systems;
– determination of criteria for choosing technical solutions;
– comparative assessment and ranking of technical solutions:
– identifying trends in the development of complex information systems, etc.

These multi-criteria tasks are difficult to formalize and therefore developers often rely on their experience in solving similar problems or informal methods of expert assessments. From formal system analysis tools, methods of mathematical statistics and operations research are usually used [8],[4]. The determination of the feasible solution set is a major challenge in engineering optimization problems. In multi-criteria problems the researcher can often identify the main variables, establish connections between them, i.e. build a model that adequately reflects the situation, but the preferred combinations of criteria cannot be determined on the basis of objective information. For selection the best solutions a compromise between various criteria is required.

In this paper we observe several approaches to solving the technical problem together with proposing the method of searching for the optimal multi-criteria ranking of alternatives on the basis of individual single-criterion rankings. The proposed research is the extended version of conference paper [18], presented at CERCIRAS 2021 Workshop. Previous paper described the basics of the problem of complex assessments of the technical solutions and the formal approach for solving this problem, including the methods from the theory of Data Mining and Operation Research. It introduced the concept of multicriteria optimization in finding the optimal variant of a multiprocessor system and in searching for a trade-off between several alternatives. The current study extends the research in this direction. Based on a simple example it demonstrates several approaches to searching and narrowing down the Pareto space, including those involving expert knowledge. The approaches for converting a multi-criteria problem to a single-criteria optimization problem by criteria convolution, as well as for ranking solutions based on distances to the ideal point are described and presented on the example. In the last section it is proposed a method based on aggregating ordered ranked lists for searching for the effective technical solutions, characterized by several system resources. The initial rankings of alternatives for each criterion are aggregated in order to find the optimal solution. In this case a single-criteria minimization problem takes into account the sum of distances of the solution candidates to the initial rankings. The proposed method provides more flexibility in selecting the best variant of solution based on several criteria. The method allows to take into account both the objective values of criteria and the decision maker preferences.

The paper notes the advantages and disadvantages of various approaches, as well as the dependence of the result on both the initial parameters of the mathematical models and the expert preferences. It is noted that the use of the methods for ranking solutions allows determining not only the best solution, which is often of primary interest, but a complete rating of all solutions, which provides information for decision-making.

2. Related work

Multi-criteria decision analysis (MCDA) is a multi-step process consisting of a set of methods to structure and formalise decision-making processes in a transparent and consistent manner. Over the years, MCDA methods and software tools are used for a large number of applications from modeling, optimization and decision-making tasks, to performance's simulation [10].

To date, a lot of research has been carried out in the field of multi-criteria decision selection [17]. They are aimed to model decision making process and require the participation of experts in reaching a decision based on many criteria [9]. The most popular are the AHP method, based on pairwise comparison of hierarchical criteria considering difference information; ANP method, which is a non-linear and more general type of AHP using Markov-chain-based aggregation; FUZZY AHP method with the fuzzy evaluation of the alternatives; ELECTE method, based on outranking the relationship of the alternatives and using pairwise comparison; PRAGMA method, which compares partial profiles of alternatives considering all the possible criteria pairs and etc.

In traditional MCDA methods often the single optimal solution is chosen by collecting the DM's preferences where multicriteria optimization (MCO) and decision-making tasks are combined for obtaining a point by point search approach [10]. The final obtained solutions must be as close to the true optimal solution as possible and the solution must satisfy the preference information.

When considering multi-criteria task from the point of view of MCO theory, a non-dominated set in the criteria space or a Pareto-efficient set in the solution space is usually considered. In multicriteria methods, a solution to an MCO problem is understood as a single point of a Pareto-efficient set that is preferable for the decision maker. Although sometimes MCO methods require finding a small number of solutions that are interesting from the decision maker's point of view [6].

Methods for solving the MCO problem in the framework of MCDA are extremely diverse [11] . There are several ways to classify these methods [17]. Considering the decision-making process and the additional information about the preferences of the decision maker the following classes can be distinguished:

– methods that do not take into account the preferences of the decision maker (no-preference methods);
– a posteriori methods;
– a priori methods;
– interactive methods.

In the first class of methods the task is to find some compromise solution, usually in the central part of the Pareto front or the construction of a scalar optimization function without the participation of decision makers. A posteriori methods involve the decision maker entering information about their preferences into the MCO system after a certain set of non-dominated solutions has been obtained. In this regard, all methods of this class at the first stage construct an approximation of the Pareto set. A priori methods are designed to overcome the main disadvantage of a posteriori methods associated with the construction of the entire reachability set. Here it is assumed that the decision maker introduces additional information about his preferences before starting to solve the problem, a priori. Most often, this information is formalized in such a way as to reduce a multi-criteria problem to a single-criteria one. Examples include the scalar convolution method, the e-constraint method [12], lexicographic ordering and goal programming [1]. Interactive methods consist of a set of iterations, each of which includes an analysis stage performed by the decision maker and a calculation stage performed by the MCO system [7]. Based on the nature of the information received by the MCO system from the decision maker at the analysis stage, classes of interactive methods can be distinguished, in which the decision maker directly assigns weighting coefficients to particular optimality criteria; imposes restrictions on the values of particular optimality criteria or evaluates the alternatives proposed by the MCO system. The selection of the appropriate MCO method for new technical solutions depends on the complexity of the decision space, existence and type of expert knowledge and available time. Each MCO method has its own definition of best alternative and it is not determined if using same input data in different methods will give the same results.

Our paper makes the overview and comparative assessment of several a posteriori and a priori approaches to the selection of best variant of technical system based on several criteria, formulating this problem as a MCO task. In the course of describing the methods and applying them using a simple example we demonstrate how similar or different results might evolve. Together with several popular approaches, the method based on aggregation of ordered ranked lists is described. It can be considered as a way of integration between MCO and MCDA processes and shows its consistency on a par with well-known methods. The method allows searching for solution based on both the available numerical criteria values and the expert preferences in ranking alternatives on several criteria.

The described approaches can be applied to the practical tasks in the area of technical system design and can be integrated as the component of a computer system that supports engineering decision-making activities.

3. Comprehensive Approach to System Design of Multiprocessor Information Systems

3.1. Formulation of the System Design Problem

The scope of application of an information system is mainly determined by both functions and technical characteristics, which are specified by numerical or nominal values and limitations. For example, the multiprocessor on-board electronics system of a modern car or robotic mobile platform has a distributed multiprocessor architecture, which allows for real-time control of units (i.e., with a given performance). To provide interprocessor interfaces, a CAN (Controller Area Network) bus is used, which provides sufficient noise immunity and reliability of communications. The number of peripheral processors is a consequence of the number of terminal devices (i.e., the required functional performance). In addition, a consequence of the second order are the parameters of power consumption and mass-dimensional characteristics. If the design domain is a distributed computing system on a chip (SoC), then the priority relationships of mutual influence will be changed and supplemented, perhaps significantly.

The composition and mutual influence of the specified characteristics (and/or parameters) of the system can be represented in the form of an undirected graph, as shown in Fig. 1. In Fig. 1, the graph reflects only qualitative dependencies. Even a quick analysis of a fragment of the original graph indicates the close interdependence of parameters in the system. Sometimes the named dependencies are obvious, for example, performance versus the number of processors, and they are easy to represent in the form of formulas. However, this is not always the case. For example, it is difficult (and perhaps impossible) to indicate the dependence of the reliability or mechanical layout (weight and dimensions) of a microprocessor system on the selected type of interface, although it is obvious to every designer that such a dependence exists. This graph is for demonstration purposes and, naturally, can be supplemented with other parameters and connections.

In general, the task of system design is to find a conceptual solution whose parameters will be within acceptable, or better yet, optimal values.

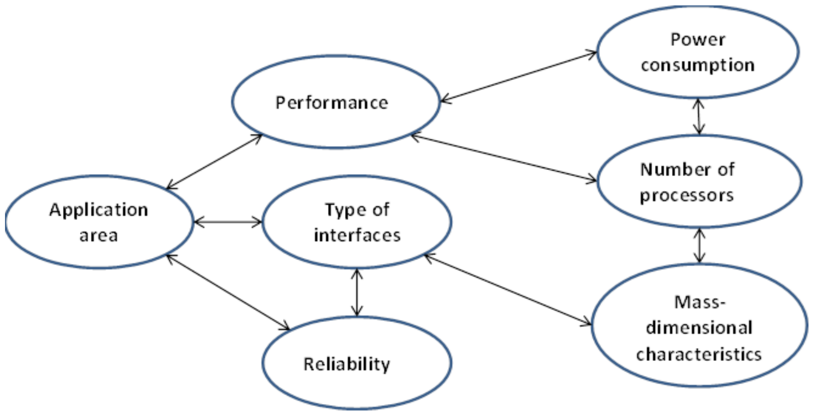


Fig. 1. Undirected graph, reflecting the subject area of system design

3.2. System Design Methodology

The proposed system design methodology is as follows:

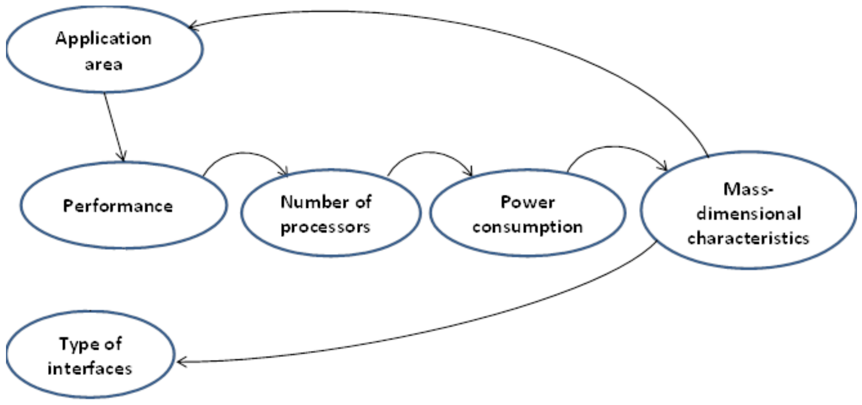


Fig. 2. A directed graph reflects one of the possible system design scenarios

1. The developer (or general designer, or system architect) draws up an initial graph, similar to Fig. 1, reflecting two-way relationships in system design in a given subject area.
2. In the undirected graph, the expert identifies key and derived parameters (the reliability parameter is not considered as a key one).
3. The original graph is modified into a directed one, and the number of mutual (less significant) connections is reduced. Ideally, there could be a chain of cause and effect relationships. In Fig. 2 on the basis of expert experience and understanding of all the restrictions available to him, a special case of relationships are considered. According to Fig. 2, to ensure a given performance it is necessary to use, for example, 6 processors, which in turn will increase energy consumption by 6 times compared to a single-processor implementation. These changes will cause an increase in weight and size characteristics and will affect the appearance of the interface.
4. The found chain is supplemented (detailed) with the necessary (known) quantitative data and restrictions. A rapid evaluation of the possibility of achieving the result is carried out using known theoretical dependencies, expert knowledge, as well as multi-criteria data analysis.
5. Very often parameters have implicit and mutually exclusive relationships. Therefore, it is important to "check feedback" during system design. For example, it may turn out that the obtained mass-dimensional characteristics are not consistent with the conceptually accepted interface, or the subject area. In this case, the system design process is iteratively repeated until an acceptable system solution is found.

4. Comparative Evaluation of Multiprocessor Systems as a Problem of Multicriteria Optimization

The greatest difficulty in designing complex multiprocessor information systems lies in the need to solve the problem of multicriteria optimization according to multiple optimization criteria, which are individual system parameters or used system resources. As a rule, the criteria are interdependent, i.e. an increase in one of them can lead to a decrease in the value for the other. Using multicriteria optimization makes it possible to identify compromise or non-dominant solutions, where none of them is better than the other in all the parameters under consideration and, therefore, are of equal importance. In this case, many solutions are allowed, each of which is acceptable in the absence of preliminary information about the importance of the criteria.

In general, the problem of multicriteria optimization is formulated as follows: Find the vector $\bar{x}^* = (x_1^*, x_2^*, \dots x_n^*)^T$ of the values of parameters, which satisfy m inequalities:

$$g_i(\bar{x}) \geq 0,\ i = 1, 2, \dots, m, \tag{1}$$

and p equalities

$$h_i(\bar{x}) = 0,\ i = 1, 2, \dots, p, \tag{2}$$

and optimize the vector function

$$f(\bar{x}) = [f_1(\bar{x}), f_2(\bar{x}), \dots, f_k(\bar{x})]^T\,. \tag{3}$$

Constraints (1), (2) define the domain G, which contains all feasible solutions to the problem. The vector $\bar{x}^*$ corresponds to the optimal solution in the domain G. In this case, the optimality is meant according to the Pareto concept, the formal definition of which from the point of view of the maximization problem is as follows [3]:

The vector of solution $\bar{x}^*$ is called Pareto-optimal if and only if there is no other vector $\bar{x}$,which dominate $\bar{x}^*$,i.e. if

$$\forall i \in 1, 2, \dots, k,\ f_i(\bar{x}) \leq f_i(\bar{x}^*)\ and\ \exists i \in 1, 2, \dots, k,\ where\ f_i(\bar{x}) < f_i(\bar{x}^*)$$

In other words, $\bar{x}^*$ is Pareto optimal if there are no acceptable vectors $\bar{x}$, which allow you to increase the value of one of the criteria, while not decreasing the value of at least one of the remaining criteria. The solution $\bar{x}^*$ is strictly dominates the solution $\bar{x}$, if $\forall i \in 1, 2, \dots, k,\ f_i(\bar{x}) < f_i(\bar{x}^*)$. In general, the solution to the optimization problem is a set of non-dominated solutions. Having a set of several non-dominated solutions obtained as a result of multicriteria optimization, it is possible to choose a solution that is most preferable for a specific applied problem.

Consider a demonstration example of solving the problem of comparative evaluation of multiprocessor systems. Table 1 presents four multiprocessor systems, which are characterized by several parameters: the number of processors, performance, dimensions, power consumption. We will assume that these systems can programmatically solve the same problem, but are implemented in a different design.

For clarity and simplicity of reasoning, we use the simplest normalization method, when the parameters are reduced to some fixed maximum value (in our case, 100, 10, 10 and 100, respectively). The normalized values of the parameters of multiprocessor systems are shown in Table 1 in the corresponding columns after the sign (/).

Table 1. Parameters of multiprocessor systems

System	Number of processors (items)	Performance (Gflops)	Dimensions (Volume, dm3)	Power consumption (W)
System 1	4/0,04	2,0/0,2	4,0/0,4	20,0/0,2
System 2	8/0,08	3,0/0,3	1,0/0,1	12,0/0,12
System 3	16/0,16	5,0/0,5	3,0/0,3	10,0/0,1
System 4	64/0,64	7,0/0,7	6,0/0,6	40,0/0,4

If the number of processors and performance are known, statistically confirmed parametric dependencies (Fig. 3a), then the dependences of dimensions and power consumption in this example are not visible (Fig. 3b). (Although, in theory, we know that with an increase in the number of processors and performance, the power consumption and size should increase). But this example deliberately does not provide any information about the technology, element base, design, purpose, generation, etc., which directly affects both the dimensions and power consumption of the implemented systems.

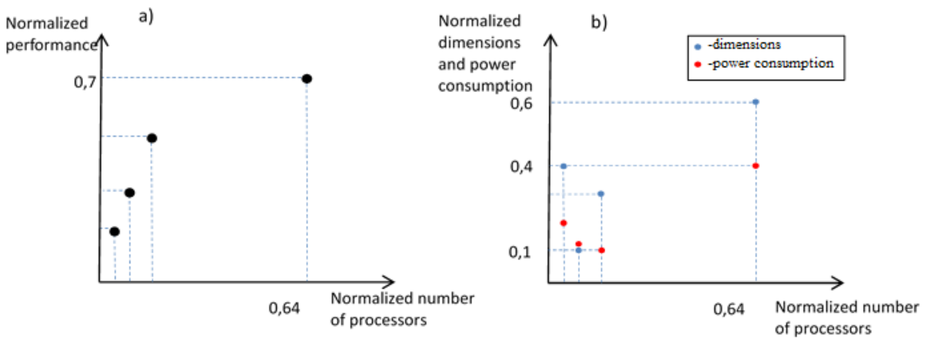


Fig. 3. Representation of multiprocessor systems in two-dimensional parameter space

Let perform the assessment of the systems presented in Table 1 based on the concept of non-dominated solutions, considering the value of each criterion separately. We will assume that it makes no sense to include the number of processors as an individual criterion, since it is indirectly in the performance of systems. In our case we will perform a manual assessment of the systems. To solve more complex problems with a large number of alternatives, characterized by a large number of parameters, special methods of multicriteria optimization are used [15], [3],[19].

In the presented example, the criteria to be optimized correspond to the parameters of the system, and the problem of multicriteria optimization in this case can be written in the following form:

Optimize vector function

$$F(x) = [x_1, x_2, x_3]$$
$$x_1 \to max, x_2 \to min,\ x_3 \to min \tag{4}$$

and $a_1 \leq x_1 \leq b_1, a_2 \leq x_2 \leq b_2, a_3 \leq x_3 \leq b_3$ – allowed parameter range, where x_1, x_2, x_3 - performance, dimensions and power consumption.

Consider Systems 1 and 2, where the value of the performance criterion of System 2 are higher than the corresponding value of System 1, and the value of the dimensions and power consumption criteria for System 2 are lower. Consequently, System 2 has more optimal parameter values and thus dominates System 1. If we compare Systems 2 and 3, then we see that System 3 has higher performance and less power consumption, while its dimensions are larger than the dimensions of System 2. Therefore, Systems 2 and 3 are non-dominated solutions. Likewise, the non-dominated solutions are Systems 2 and 4, Systems 3 and 4.

In Fig. 4 the systems under consideration are shown as points in the space of two criteria. Non-dominated solutions are connected with a line.

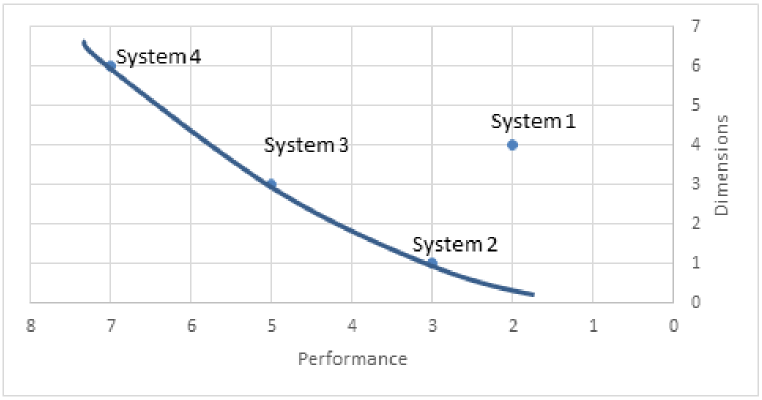


Fig. 4. Graphic presentation of technical solutions in a two-criteria space

It should be noted that Systems 2, 3, 4 are not dominant in the specific problem under consideration, i.e. they are locally non-dominated solutions. In the case of adding additional alternative solutions for multiprocessor systems, the number and composition of non-dominated solutions may change, although in this case the solutions will be locally non-dominated. According to the analysis System 1 can be excluded from further consideration due to the fact that it is the dominant solution for all the considered optimization criteria.

To further narrow down the Pareto set and identify the single best solution, some additional information is needed [14]. One of the main types of additional information, often used when solving various multicriteria problems, is information about the comparative importance of partial criteria, which is usually given in the form of numerical coefficients $w_k \geq 0$, characterizing the importance of partial criterion $f_i, i = 1, \ldots, k$. The importance coefficients together constitute a weight vector $w = (w_1, \ldots, w_k)$, the components of which are usually normalized by the condition $\sum_k w_k = 1$. The most well-known methods for calculating coefficients include sequential comparison of criteria by importance, pairwise comparison of criteria by absolute or relative importance.

Using additional information about the importance of criteria, you can narrow down the set of non-dominated solutions. Consider two non-dominated solutions $y_1 = (y_{11}, \ldots, y_{1k})$ and $y_2 = (y_{21}, \ldots, y_{2k})$, where $y_{1p} > y_{2p}, y_{1q} < y_{2q}$ and $y_{1h} = y_{2h}$ for all $h \neq p, q$. Formally, these alternatives are incomparable in terms of dominance. However, for the decision maker (DM), alternative y_1 is preferable to y_2. This means that the p-th partial criterion is more important than the q-th partial criterion with parameters $r_p = y_{1p} - y_{2p} > 0, r_q = y_{2q} - y_{1q} > 0$. When choosing one of two solutions, the DM agrees to lose the value r_q according to a less important criterion in order to receive an additional gain r_p according to a more important criterion.

The number $t_{pq} = \frac{r_q}{(r_p + r_q)}$ is called the proportional coefficient of relative importance for the p-th and q-th criteria. At $t_{pq} = 0,5$, the values of losses and gains coincide. Thus, the set of selected solutions is contained in the restricted Pareto boundary, consisting of vectors $y' = f'(x) = (f'_1(x), \ldots, f'_k(x))$ with components determined by the expressions:

$$f'_q(x) = t_{pq} f_p(x) + (1 - t_{pq}) f_q(x);\ f'_h(x) = f_h(x), \forall h \neq q \tag{5}$$

According to (5), new values of the criteria vector are obtained from the previous ones by replacing the less important criterion $f_q(x)$ with a convex combination of criteria $f_p(x)$ and $f_q(x)$.

Consider the application of the above-mentioned method of narrowing the Pareto space using the example described in Table 1. Let us consider three previously obtained non-dominated solutions, represented by vectors of three criteria values $S_2 = (0,3; -0,1; -0,12)$, $S_3 = (0,5; -0,3; -0,1)$ and $S_4 = (0,7; -0,6; -0,4)$, where the values of the last two criteria are inverted in order to bring them to the maximization problem.

Let us assume that the DM considers the first criterion f_1 more important than the second f_2 with a proportional coefficient of relative importance $t_{12} = 0,8$. Recalculate the second component of each vector $S_i, i = 2, \ldots, 4$ using the formula $f'_2(S_i) = 0,8 f_1(S_i) + 0,2 f_2(S_i)$. We obtain new vectors $S'_2 = (0,3; 0,22; -0,12)$, $S'_3 = (0,5; 0,34; -0,1)$ and $S'_4 = (0,7; 0,44; -0,4)$. It is obvious that solution S'_3 dominates solution S'_2. Therefore, in the narrowed Pareto space there will remain two solutions S'_3 and S'_4, which correspond to System 3 and System 4.

5. Approaches to Converting a Multi-Criteria Problem to a Single-Criteria One

One of the ways to solve a multi-criteria problem for selection a technical solution is to transform it into a single-criteria one by combining all partial criteria $f_j(x), j = 1, \ldots, k$ into one general quality criterion $f(x) = F(f_1(x), f_2(x), \ldots, f_k(x))$, which is otherwise called criteria convolution. Then the search for the best solution reduces to finding the extremum of the single function $f(x)$

$$x^* \in argmax_{x \in X^a} f(x), \tag{6}$$

where x is an alternative from the acceptable set X^a.

As a rule, weighted convolutions of partial criteria for the effectiveness of a technical solution are used

$$f(x) = \sum_{j=1}^{k} w_j f_j(x)$$
$$f(x) = \prod_{j=1}^{k} w_j f_j(x) \; f(x) = \prod_{j=1}^{k} [f_j(x)]^{w_j}, \tag{7}$$

where $w_j \geq 0$ is a weight of the partial criterion $f_j(x)$.

Partial performance criteria are usually normalized in one of the following ways:

$$f_j'(x) = \frac{f_j(x)}{y_j^{max}},\; f_j'(x) = \frac{f_j(x)}{(y_j^{max} - y_j^{min})},\; f_j'(x) = \frac{f_j(x) - y_j^{min}}{(y_j^{max} - y_j^{min})} \tag{8}$$

where y_j^{min}, y_j^{max} – minimal and maximal values of the partial criterion $f_j(x)$.

The choice of the weights of the partial criteria in the optimization function (7) can influence the result of the optimal solution and is usually set by the DM by a number of well-known methods [13]. Consider the implementation of the above approach for the example from Table 1. The complex criterion for the comparative assessment of systems is a weighted sum of three criteria. In this case, it is necessary to perform the following steps:

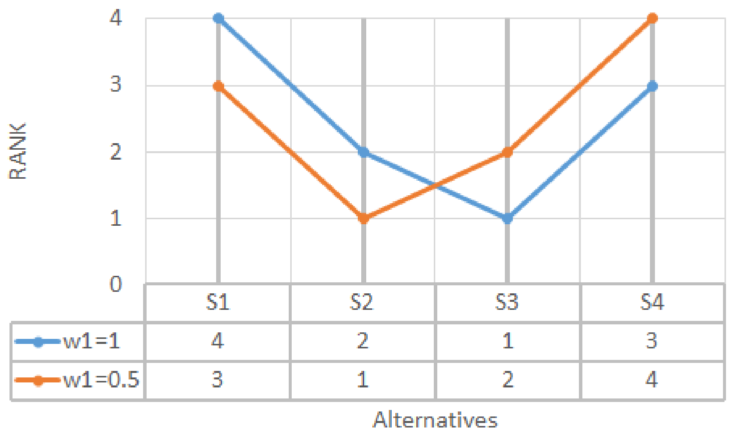
1. Bringing the criteria to a single range of values (normalization or scaling of the criteria):
$$\bar{x_i} = \frac{x_i - x_i^{min}}{x_i^{max} - x_i^{min}},$$
where x_i^{min} is the minimal value of i-th criterion, x_i^{max} is the maximal value of i-th criterion. For our example $x_i^{min} = 0$ for $i = 1, 2, 3$ and $x_1^{max} = 10$, $x_2^{max} = 10$, $x_3^{max} = 100$.
2. Selection of weight coefficients w_1, w_2, w_3 for each criterion: performance, dimensions and power consumption.
3. Ranking solutions according to a complex criterion $F(x) = -w_1\bar{x_1} + w_2\bar{x_2} + w_3\bar{x_3}$, where the value of w_1 is taken with minus sign to convert to the minimization problem according to the first criterion.

The minimum value of the complex criterion corresponds to the most optimal solution. The normalized values of the system parameters are presented in Table 1. Let us choose the following values of the weight coefficients $w_1 = w_2 = w_3 = 1$. Let's calculate the values of the complex criterion for each of the systems $S_1 = (0,2; 0,4; 0,2)$, $S_2 = (0,3; 0,1; 0,12)$, $S_3 = (0,5; 0,3; 0,1)$ and $S_4 = (0,7; 0,6; 0,4)$

$$F_1(x) = -1 * 0,2 + 1 * 0,4 + 1 * 0,2 = 0,4$$
$$F_2(x) = -1 * 0,3 + 1 * 0,1 + 1 * 0,12 = -0,08$$
$$F_3(x) = -1 * 0,5 + 1 * 0,3 + 1 * 0,1 = -0,1$$
$$F_4(x) = -1 * 0,7 + 1 * 0,6 + 1 * 0,4 = 0,3.$$

Thus, the systems presented in Table 1 according to the complex criterion can be ranked as follows System 3, System 2, System 4 and System 1 in ascending order of the criterion value. The most optimal system is System 3.

The disadvantage of this ranking method is the dependence of the optimal solution on the choice of the values of the weight coefficients w_1, w_2, w_3. For example, if the value of the weight $w_1 = 0.5$, then System 2 will be selected as the most optimal (Fig. 5).



	S1	S2	S3	S4
w1=1	4	2	1	3
w1=0.5	3	1	2	4

Fig. 5. Ranking of alternatives according to a complex criterion $F(x)$ with different values of weight coefficient w_1

Another approach for solving a multicriteria problem is to search for alternatives with given characteristics. In this case, you can indicate the values of partial quality criteria $y_j^0, j = 1, \dots, k$ that are desirable for the DM or the boundaries of their change. The set of such values $y = (y_1^0, \dots, y_k^0)$ is called the reference point. Two characteristic reference points are, in particular, $y^{max} = (y_1^{max}, \dots, y_k^{max})$ and $y^{min} = (y_1^{min}, \dots, y_k^{min})$, where y_i^{min} is the minimal value of i-th criterion, y_i^{max} is the maximal value of i-th criterion.

To solve a multicriteria optimization problem, a search is made for an alternative or technical solution that is closest to the reference point. In the multidimensional space of evaluations based on partial quality criteria, a certain proximity measure $d[f(x), y^0]$ is specified between the points $f(x) = (f_1(x), f_2(x), \dots, f_k(x))$ and $y^0 = (y_1^0, \dots, y_k^0)$, where $f(x)$ is the alternative and y^0 is the reference point. Then the optimal solution is defined as

$$x^* \in argmin_{x \in X^a} d[\, f(x), y^0] \tag{9}$$

Usually one of the metrics of the k-dimensional vector space (R^k, d_p) is chosen as a proximity measure, such as the weighted Euclidean metric

$$d_2[f(x), y^0] = [\sum_{j=1}^{m} w_j (f_j(x) - y_j^0)^2]^{\frac{1}{2}}, \tag{10}$$

where w_j is the coefficient of importance of the partial criterion $f_j(x)$.

Setting one or another proximity measure $d[f(x), y^0]$ is another possible way of convolving partial criteria and transforming a multicriteria problem into a single-criteria one. The disadvantage is that different metrics $d[f(x), y^0]$ may correspond to different optimal options for the technical solution x^*. Despite the disadvantage, this method for solving a multicriteria problem is widely used in practice [13].

One example of the implementation of multicriteria optimization taking into account reference points is the TOPSIS (Technique for Order Preference by Similarity to Ideal Solution) method proposed in [5]. The main idea of the method is as follows: after determining the "ideal" or best and "ideal-negative" or worst expected states (alternatives), an attempt is made to find a solution that would allow one to get as close as possible to the "ideal" state and remain as far away from the "ideal-negative" state. The decision-making process begins with the evaluation of all alternative solutions according to all criteria. As a result, a decision matrix is formed. The method consists of six consecutive steps: 1) calculation of the normalized decision matrix; 2) calculation of a weighted normalized decision matrix; 3) definition of the "ideal" and "ideal-negative" expected state; 4) calculation of the metric values; 5) calculation of relative proximity to the "ideal" state; 6) ranking of alternatives.

Mathematical description of the TOPSIS method is as follows:

Step 1. Formation of a matrix of assessments or decisions $(x_{ij})_{n \times k}$, consisting of n alternatives and k criteria, where the element of the matrix x_{ij} determines the value of the j-th criterion of the i-th alternative.

Step 2. By normalizing the matrix $(x_{ij})_{n \times k}$ forming the matrix $R = (r_{ij})_{n \times k}$ as follows:

$$r_{ij} = \frac{x_{ij}}{\sqrt{\sum_{t=1}^{n} x_{tj}^2}},\ i = 1, 2, \ldots, n;\ j = 1, 2, \ldots, k. \tag{11}$$

Step 3. Calculation of the weighted normalized decision matrix:

$$t_{ij} = r_{ij} \cdot w_j,\ i = 1, 2, \ldots, n;\ j = 1, 2, \ldots, k, \tag{12}$$

where w_j – weight coefficient of j-th criterion, $j = 1, 2, \ldots, k$ and $\sum_{j=1}^{k} w_j = 1$.

Step 4. Determination of the worst A_w and best A_b alternative:

$$\begin{aligned} A_w = \{&\langle max(t_{ij}|i = 1, 2, \ldots, n)|j \in J_-\rangle, \\ &\langle min(t_{ij}|i = 1, 2, \ldots, n)|j \in J_+\rangle\} \equiv \{t_{wj}|j = 1, 2, \ldots, k\}, \\ A_b = \{&\langle min(t_{ij}|i = 1, 2, \ldots, n)|j \in J_-\rangle, \\ &\langle max(t_{ij}|i = 1, 2, \ldots, n)|j \in J_+\rangle\} \equiv \{t_{bj}|j = 1, 2, \ldots, k\}, \end{aligned} \tag{13}$$

According to (13) the value of an individual criterion of the worst alternative A_w is equal to the maximum value of this criterion for the alternatives under consideration in the case of minimizing the criterion ($j \in J_-$) and equal to the minimum value of this criterion for the alternatives under consideration in the case of maximizing the criterion ($j \in J_+$). And the reverse reasoning applies to the best alternative A_b.

Step 5. Calculation of L^2-distance between target alternative i and the worst alternative A_w

$$d_{iw} = \sqrt{\sum_{j=1}^{k} (t_{ij} - t_{wj})^2},\ i = 1, 2, \ldots, n, \tag{14}$$

and the distance between i-th alternative and the best alternative A_b

$$d_{ib} = \sqrt{\sum_{j=1}^{k} (t_{ij} - t_{bj})^2}, \; i = 1, 2, \ldots, n, \tag{15}$$

where d_{iw} and d_{ib} are distances according to the L^2-norm.

Step 6. Calculation of relative proximity to the ideal solution:

$$s_{iw} = \frac{d_{iw}}{(d_{iw} + d_{ib})}, \; 0 \le s_{iw} \le 1, \; i = 1, 2, \ldots, n. \tag{16}$$

The value $s_{iw} = 1$ if and only if the alternative corresponds to the best state; and $s_{iw} = 0$ if and only if the alternative corresponds to the worst state.

Step 7. Ranking alternatives by values s_{iw} $(i = 1, 2, \ldots, n)$.

Let's consider the implementation of the TOPSIS method for the example from Table 1. Let us choose the following values of the weight coefficients $w_1 = w_2 = w_3 = 1$. According to (12) the values of the weighted normalized decision matrix $t_{ij}, \; i = 1, 2, 3, 4; \; j = 1, 2, 3$ for alternatives $S_i = t_{ij}, \; i = 1, 2, 3, 4$ are the following $S_1 = (0,2; 0,4; 0,2)$, $S_2 = (0,3; 0,1; 0,12)$, $S_3 = (0,5; 0,3; 0,1)$ and $S_4 = (0,7; 0,6; 0,4)$. According to (13) we define the best alternative as

$$A_b = (\max(0,2; 0,3; 0,5; 0,7); \min(0,4; 0,1; 0,3; 0,6); \min(0,2; 0,12; 0,1; 0,4)) = (0.7; 0.1; 0.1)$$

and the worst alternative as

$$A_w = (\min(0,2; 0,3; 0,5; 0,7); \max(0,4; 0,1; 0,3; 0,6); \max(0,2; 0,12; 0,1; 0,4)) = (0,2; 0,6; 0,4).$$

The graphical representation of the choice of the best A_b and worst A_w alternatives is shown in Fig. 6.

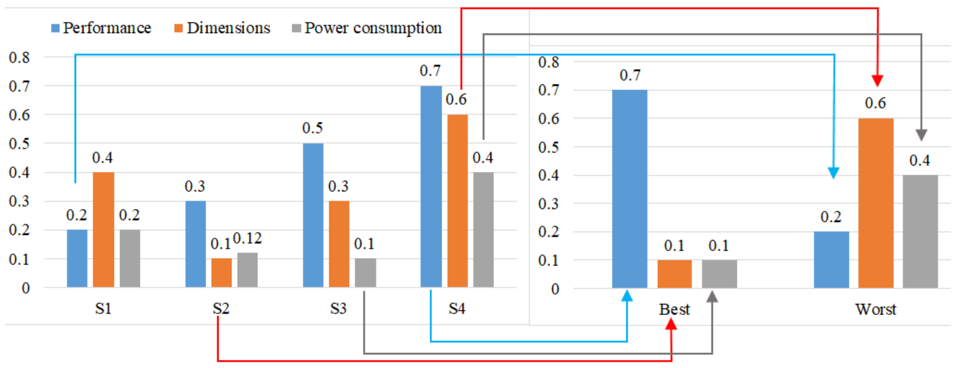


Fig. 6. Illustration of the selection of the best and worst alternatives

According to (14) and (15) L^2-distances between alternatives $S_i,\ i = 1, 2, 3, 4$ and the worst and best alternatives A_w and A_b are

$$d_{1b} = \sqrt{(0,2-0.7)^2 + (0,4-0.1)^2 + (0,2-0.1)^2} = \sqrt{0,25+0,09+0,01} = \sqrt{0,35} \approx 0,2828$$

$$d_{1w} = \sqrt{(0,2-0,2)^2 + (0,4-0,6)^2 + (0,2-0,4)^2} = \sqrt{0+0,04+0,04} = \sqrt{0,08} \approx 0,5916$$

$$d_{2b} = \sqrt{(0,3-0.7)^2 + (0,1-0.1)^2 + (0,12-0.1)^2} = \sqrt{0,16+0+0,0004} = \sqrt{0,1604} \approx 0,4005$$

$$d_{2w} = \sqrt{(0,3-0,2)^2 + (0,1-0,6)^2 + (0,12-0,4)^2} = \sqrt{0,01+0,25+0,0784} = \sqrt{0,3384} \approx 0,5817$$

$$d_{3b} = \sqrt{(0,5-0.7)^2 + (0,3-0.1)^2 + (0,1-0.1)^2} = \sqrt{0,04+0,04+0} = \sqrt{0,08} \approx 0,2828$$

$$d_{3w} = \sqrt{(0,5-0,2)^2 + (0,3-0,6)^2 + (0,1-0,4)^2} = \sqrt{0,09+0,09+0,09} = \sqrt{0,27} \approx 0,5196$$

$$d_{4b} = \sqrt{(0,7-0.7)^2 + (0,6-0.1)^2 + (0,4-0.1)^2} = \sqrt{0+0,25+0,09} = \sqrt{0,34} \approx 0,5831$$

$$d_{4w} = \sqrt{(0,7-0,2)^2 + (0,6-0,6)^2 + (0,4-0,4)^2} = \sqrt{0,25+0+0} = \sqrt{0,25} \approx 0,5$$

After determining the distances from each alternative to the positive and negative ideal solutions, the values of relative proximity to the ideal solution s_{iw} are calculated according to (16) and presented in Table 2. According to the values of the relative similarity s_{iw} in Table 2 the systems are ranked as in the last column and the most optimal system is System 3.

Table 2. Alternatives ranking results

Alternative	d_{iw}	d_{ib}	s_{iw}	Rank
S_1	0,2828	0,5916	0,3234	4
S_2	0,5817	0,4005	0,5922	2
S_3	0,5196	0,2828	0,6475	1
S_4	0,5	0,5831	0,4616	3

6. Ranking Technical Solutions by Aggregating Ordered Ranked Lists

In addition to known methods in our study we consider another approach for converting a multicriteria optimization problem to a single-criteria representation when searching for technical solutions. The method is based on ranking individual alternatives for each optimization criterion separately, followed by aggregation of ordered ranked lists [2],[16].

Let there be n alternatives $S_i,\ i = 1, \dots, n$, each of which is characterized by the values of k indicators (criteria) $S_i = (x_{i1}, \dots, x_{ik})$, and the indicators can be both quantitative and qualitative. The values of the criteria belong to the set $X^a \subseteq X = X_1 \times \cdots \times X_k$ of acceptable values. Heterogeneous values of indicators are usually transformed to a single measurement scale.

According to the method, all alternatives are initially ranked for each of the k criteria, where L_i is a list of alternatives ordered relative to the values of the i-th criterion. Thus, the following single-criteria minimization problem is formulated:

$$\delta^* = argmin\ \Phi(\delta), \tag{17}$$

where $\Phi(\delta) = \sum_i d(\delta, L_i)$, d is the distance function, and minimization is carried out with respect to all possible ordered lists δ of dimension $n = |L_i|$.

Selecting a suitable distance function d is one of the most important steps of the method. Two options were chosen as such functions: Spearman distance and Kendall rank distance. The Spearman distance between an ordered list of alternatives L_i and any ordered list δ can be defined as

$$S(\delta, L_i) = \sum_{t \in L_i \cup \delta} |r^{\delta}(t) - r^{L_i}(t)|, \tag{18}$$

where $r^{L_i}(t)$ – rank of alternative t in the list L_i.

The smaller the value of the metric $S(\delta, L_i)$, the more similar the two lists. To take into account additional information about the values of the criteria for each alternative a weighted Spearman distance is determined

$$WS(\delta, L_i) = \sum_{t \in L_i \cup \delta} |M(r^{\delta}(t)) - M(r^{L_i}(t))| \cdot |r^{\delta}(t) - r^{L_i}(t)|, \tag{19}$$

where $M(A)$ – value of the i-th criterion of alternative A.

Kendall's rank distance is a metric that counts the number of pairwise divergences between two ranked sets of feature values. The greater this distance, the more different the two characteristics are, and, therefore, the less the dependence between them.

In the case when the number of alternatives is small, the single-criteria problem (17) can be solved by simply searching through the variants of ordered lists and choosing the alternative located at the top position of the list as the optimal solution.

So, for example from Table 1, we can create three ordered lists according to three optimization criteria (Table 3), where the last column presents the optimal ranking δ corresponding to the minimum of the functional $\Phi(\delta) = 8$.

In order to select the optimal ranking δ, we have applied the simple strategy of calculating the $\Phi(\delta)$ value for all possible permutations of the rankings of alternatives $S_i,\ i =$

Table 3. Ranked lists of alternatives

Alternative	Performance	Dimensions	Power consumption	Optimal rank
S_1	4	3	3	3
S_2	3	1	2	2
S_3	2	2	1	1
S_4	1	4	4	4

$1, 2, 3, 4$ from Table 1. After that the ranking δ corresponding to the minimum value of the function $\Phi(\delta)$ is considered as optimal. The $\Phi(\delta)$ values for each ranking of alternatives are presented in Table 4.

Table 4. Functional values for all possible rankings

No	Ranking	Value	No	Ranking	Value
1	(1 2 3 4)	16	**13**	**(3 1 2 4)**	**8**
2	(1 2 4 3)	20	14	(3 1 4 2)	16
3	(1 3 2 4)	14	**15**	**(3 2 1 4)**	**8**
4	(1 3 4 2)	22	16	(3 2 4 1)	16
5	(1 4 2 3)	18	17	(3 4 1 2)	14
6	(1 4 3 2)	22	18	(3 4 2 1)	14
7	(2 1 3 4)	14	19	(4 1 2 3)	10
8	(2 1 4 3)	18	20	(4 1 3 2)	14
9	(2 3 1 4)	12	21	(4 2 1 3)	10
10	(2 3 4 1)	20	22	(4 2 3 1)	14
11	(2 4 1 3)	16	23	(4 3 1 2)	12
12	(2 4 3 1)	20	24	(4 3 2 1)	12

In Table 4 the values in the column ”Value” is calculated as

$$\Phi(\delta) = S(\delta, L_1) + S(\delta, L_2) + S(\delta, L_3), \tag{20}$$

where $L_i, \ i = 1, 2, 3$ are the rankings of alternatives for criteria ”Performance”, ”Dimensions” and ”Power consumption” respectively, and $S(\delta, L_i)$ is a Spearman distance in (18). For example for ranking No. 1 in Table 4 and using single-criteria rankings in Table 3 the Spearman distances are calculated as

$$S(\delta, L_1) = |1-4| + |2-3| + |3-2| + |4-1| = 3+1+1+3 = 8$$
$$S(\delta, L_2) = |1-3| + |2-1| + |3-2| + |4-4| = 2+1+1+0 = 4$$
$$S(\delta, L_3) = |1-3| + |2-2| + |3-1| + |4-4| = 2+0+2+0 = 4$$

and the value of function in (20) is $\Phi(\delta) = 8+4+4 = 16$.

In Table 4 the minimal functional value $\Phi(\delta) = 8$ corresponds to the ranking No. 15: System 3, System 2, System 1 and System 4. The most optimal system is System 3. The same value of functional $\Phi(\delta)$ also corresponds to the ranking No.13: System 3, System 1, System 2 and System 4. In case of ambiguity, it makes sense to take into account the real criteria values and estimate the optimal ranking according to (19).

Using Multidimensional Scaling (MDS) to reduce the dimensionality of data the two-dimensional representation of all possible rankings from Table 4 and single-criteria rankings from Table 3 is shown in Fig. 7. In Fig. 7 three single-criteria rankings are marked with red circles and it is clearly seen that the rankings No.15 and No.13, marked in bold have a minimum total distance to three single-criteria rankings compared to other alternatives.

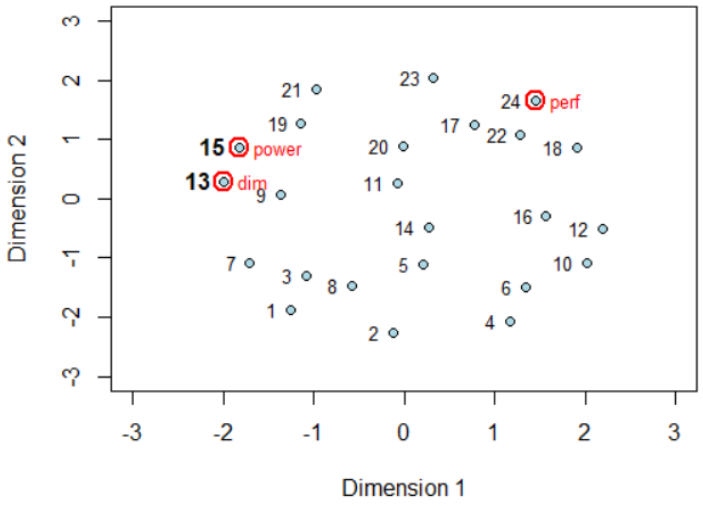


Fig. 7. Graphical representation of all alternative rankings highlighting the optimal ones

In the case of a larger number of alternatives and criteria, it is proposed to solve the optimization problem (17) using a genetic algorithm (GA) [15].

The GA consists of the following steps:

Step 1. $popSize$ ordered lists of dimension n (the number of alternatives) are randomly initialized, which form the initial set of possible solutions to the problem. The size of the population must be proportional to the number of alternatives and the number of unique elements in the original ordered lists $L_i,\ i = 1, 2, \ldots, k$.

Step 2. Depending on which distance is used, calculate the objective function for each element of the population. Then randomly select individuals from the population for the next generation of GA using weighted random sampling, where the weights are determined according to the values of the fitness (objective) function.

Step 3. Carrying out crossover operations with probability p_{cross} (transition probability), i.e. two randomly ordered lists can exchange their parts, which start from a random position with probability p_{cross}.

Step 4. The crossover operation only allows the mixing of ordered lists, but to obtain radically new solutions it is necessary to use mutations. Mutation operations are performed with probability p_{mut} (mutation probability). Thus, any list in a population can randomly change one or more of its elements.

Step 5. The algorithm stops if the optimal list does not change for successive several generations of the GA or the maximum number of iterations (GA epochs) is reached.

The advantage of using the described method to solve the problem of selecting the optimal technical solution is to obtain a complete rating of technical solutions based on their effectiveness, assessed by several criteria. Any number of alternatives with arbitrary combinations of GA parameters and/or distance functions can be considered. In addition, the researcher can decide how many performance criteria to use to obtain a reliable assessment of solutions.

7. Conclusion

This paper presents the results of extended research in the field of optimization of technical solutions according to many criteria. The problem statement and system design methodology are described in general terms. The problem of finding a technical solution, which is characterized by various parameters or used system resources, is presented as a multicriteria optimization problem that allows one to find Pareto-optimal solutions. An approach for narrowing the solution space in order to reduce the uncertainty associated with multi-criteria selection and find the optimal technical solution is described and demonstrated by example. The approaches for converting a multi-criteria selection problem to a single-criteria optimization problem by criteria convolution, as well as for ranking solutions based on calculating distances to the ideal point are described and presented on the example. It is proposed to apply a method based on aggregating ordered ranked lists for converting a multicriteria optimization problem to a single-criteria representation when searching for technical solutions. A method is based on solving a single-criteria optimization problem, where the optimization functional estimates the sum of the distances of the searched optimal ranking of alternatives to the corresponding rankings for each of the criteria under consideration.

The approaches described in the paper can be used to find a compromise between various characteristics of a technical solution in order to select the optimal variant when designing multiprocessor information systems. As a result of applying methods for ranking solutions, it will be possible to determine not only the best solution, which is often of primary interest, but a complete rating of all solutions, which provides information for further research. The practical results of applying different approaches are demonstrated using a simple example.A further direction of research is the analysis and application of interactive methods for searching for the optimal technical solution when developing a multiprocessor information system.

Acknowledgments. This work is partially supported by EU COST action CA19135, titled ”Connecting Education and Research Communities for an Innovative Resource Aware Society (CERCIRAS)”.

References

1. Colapinto, C., Jayaraman, R., Marsiglio, S.: Multi-criteria decision analysis with goal programming in engineering, management and social sciences: a state-of-the art review. Annals of Operations Research 251, 7–40 (2017)
2. Datta, S., Pihur, V., Datta, S.: An adaptive optimal ensemble classifier via bagging and rank aggregation with applications to high dimensional data. BMC bioinformatics 11(427), 1–11 (2010)
3. Deb, K.: Multi-objective optimization using evolutionary algorithms, vol. 16. John Wiley & Sons (2001)
4. Driscoll, P., Parnell, G., Henderson, D.: Decision making in systems engineering and management. John Wiley & Sons (2022)
5. Halicka, K.: Technology selection using the topsis method. Foresight and STI Governance 14(1), 85–96 (2020)
6. Karami, F., Dariane, A.: A review and evaluation of multi and many-objective optimization: Methods and algorithms. Global Journal of Ecology 7(2), 104–119 (2022)
7. Kasimoglu, F., et al.: A survey on interactive approaches for multi-objective decision making problems. J. Defense Sci. 15(1), 231–255 (2016)
8. Kochenderfer, M., Wheeler, T., Wray, K.: Algorithms for Decision Making. MIT press (2022)
9. Linkov, I., et al.: Multi-criteria decision analysis: case studies in engineering and the environment. CRC Press (2021)
10. Mosavi, A.: A multicriteria decision making environment for engineering design and production decision-making. International Journal of Computer Applications 69(1), 26–38 (2013)
11. Nabavi, S., Wang, Z., Rangaiah, G.: Sensitivity analysis of multi-criteria decision-making methods for engineering applications. Industrial & Engineering Chemistry Research 62(17), 6707–6722 (2023)
12. Nikas, A., et al.: A robust augmented e-constraint method (augmecon-r) for finding exact solutions of multi-objective linear programming problems. Operational Research 22(2), 1291–1332 (2022)
13. Petrovsky, A.: Decision Making Theory, vol. 312. M.: Publishing center "Academy", Moscow, Russia (2009)
14. Petrovsky, A.: Group Verbal Decision Analysis: Theory and Applications. Springer Nature (2023)
15. Rahimi, I., et al.: A comparative study on evolutionary multi-objective algorithms for next release problem. Applied Soft Computing 144, 110472 (2023)
16. Sekula, M., Datta, S., Datta, S.: Optcluster: an r package for determining the optimal clustering algorithm. Bioinformation 13(3), 101 (2017)
17. Taherdoost, H., Madanchian, M.: Multi-criteria decision making (mcdm) methods and concepts. Encyclopedia 3(1), 77–87 (2023)
18. Tatur, M., Novoselova, N., V., P.: Resource-aware approach in the design of complex information systems as a problem of multicriteria optimization. In: Proceedings of the CERCIRAS WS01: 1st Workshop on Connecting Education and Research Communities for an Innovative Resource Aware Society. pp. 1–6. Novi Sad, Serbia (2021)
19. Zitzler, E., Thiele, L.: Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. IEEE transactions on Evolutionary Computation 3(4), 257–271 (1999)

Mikhail Tatur is currently a professor at the Computer Department of Belarusian State University of Informatics and Radioelectronics (BSUIR), Minsk, Belarus. He received the

DrSci degree in Computer Sciences in BSUIR in 2011. His research interests include artificial intelligence, machine learning, robotics, parallel computing, technical diagnostics. He conduct research in the field of creating the complex intelligent systems and mobile robotics. He is an author of more than 160 publications in these areas.

Natalia Novoselova is currently a senior researcher at the Laboratory of Bioinformatics, United Institute of Informatics Problems (UIIP), National Academy of Sciences of Belarus (NASB), Minsk, Belarus. She received the PhD degree in Computer Sciences from the UIIP NASB in 2008. Her research interests include computational intelligence, machine learning, and bioinformatics. She conducts research in the field of creating intelligent methods for solving problems of preprocessing, exploratory analysis, and classification of multidimensional medical and biological data. She is an author of more than 50 research publications in these areas.

Marina Lukashevich received the Ph.D. degree in computer science. She is currently an Associate Professor in Belarusian State University. She is also a Postdoctoral Researcher with Belarusian State University of Informatics and Radioelectronics. Her main research interests include machine learning, computer vision and natural language processing. She conductes research in the field of one-class classification and working with unbalanced data. She is an author of more than 60 publications in these areas.

Received: July 25, 2024; Accepted: October 23, 2024.

Computer Science and Information Systems 22(2):459–490 https://doi.org/10.2298/CSIS240722022C

Research on Problem Formulations in Resource-aware Problems Across Scientific Domains and Applications

Paweł Czarnul and Mariusz Matuszek

Faculty of Electronics, Telecommunications and Informatics
Gdańsk University of Technology, Narutowicza 11/12
80-233 Gdańsk, Poland
pczarnul@eti.pg.edu.pl
mrm@eti.pg.edu.pl

Abstract. In this paper we conducted thorough analysis of research papers focused on resource aware problems and using one of the following formulations: integer linear programming (ILP), greedy algorithms (GrA), dynamic programming (DP), evolutionary algorithms (EA) and machine learning (ML). Basing on such general problem formulations we identified actual research tasks considered in many different domains. Furthermore, we analyzed each of these problems in terms of: resources being considered/subject to optimization, specific optimization algorithms, if applicable, and domains. Finally, based on over 170[1] research papers, we assessed which particular resources like: time, cost, energy, human, computer, natural resources, data/information are used in which problems formulations, which formulations and resources are used and considered in which application/domains. It can serve as reference for algorithms in particular domains or, conversely, looking for unexplored approaches in specific contexts.

Keywords: resource aware problems, resource, domain, integer linear programming, greedy approach, dynamic programming, evolutionary algorithm, machine learning.

1. Introduction and Motivation

Research in various domains is inevitably linked with specific resources as well as optimization problems. Such optimization problems are typically expressed as multi-objective optimization that involves metrics referring to the given domain, in particular resources in a given domain. We can distinguish physical resources such as computers, interconnects, cooling systems, human resources in a cloud computing center as well as more general resources such as time, energy, budget etc. We shall note that in optimization problems certain metrics are often linked to particular physical, problem specific resources e.g.: performance or power consumption of a computer node. These, in turn, can be reflected in metrics describing such a resource, i.e., execution time and energy used within a particular period. These can then be used in a multi-objective optimization. We shall note that optimization often involves trade-offs, e.g., performance vs energy [36,45], performance vs security [120], performance vs storage [79], performance vs memory [18,13], performance vs ease of programming/development effort [84].

[1] the total number of over 190 citations includes also references to related work.

While researching the topic of resource aware optimization we observed that in the literature there are several review papers considering specific resources within a particular domain. These include, for example:

- renewable energy [8,122]
- human resources management [69,23,59],
- computer systems, e.g., cloud computing [68,4],
- telecommunication [152],
- education [180],
- natural resources management [138,22],
- tourism [118,55,151],
- manufacturing [132],
- health [73,158],
- transport [115],
- space [117],
- disaster management [20,3].

We also identified some research papers on multidisciplinary (design) optimization, e.g., [37]. On the hand, to the best of our knowledge, there is no research on applicability of specific optimization problem formulations across various domains, with consideration of resources and metrics.

In this paper, we aim at conducting cross-domain analysis of research works that involve resource aware problems, in terms of resources / metrics considered, problem formulations and domains they target.

This paper is a very significantly extended version of workshop paper [39] that extends it in the following aspects:

1. Considering a new set of research works fetched from a reliable scientific database – Scopus. While the former paper considered approximately 70 works, we have now considered more than 190 research papers.
2. Involving other problem formulations such as a more general evolutionary algorithm concept (versus genetic algorithms considered before) as well as the popular and important machine learning.
3. Final classification of the research versus a larger number of resources: 8 vs 7 as well as applications/domains: 15 vs 8, for a more thorough analysis.

The outline of the paper is as follows. Section 2 details the methodology we used for selection of research papers used as input for subsequent analysis. Section 3 contains analysis of identified resource aware problems across domains with identification of resources, metrics and problem formulations. Section 4 includes comprehensive analysis of the previous problem descriptions with cross linking resources and problem formulations, applications/domains and problem formulations as well as resources and domains. Finally, Section 5 contains summary and outline of possible future work.

2. Methodology for Selection of Source Scientific Works

In this paper we build on and significantly extend the results originally obtained in paper [39]. In that work, analysis was based on selected scientific papers found by the standard Google search engine returned for querying for combinations of a given problem formulation and phrases: *resource*, *resource-aware problems*. The original problem formulations included: integer linear programming, dynamic programming, greedy approach as well as genetic algorithm. Furthermore, this input data set has been extended with selected results returned by the Bing search engine, queried about *resource aware computing* and *resource aware computing problems*.

In this paper, we significantly extended our previous input data set by adding scientific papers returned by the Scopus database. We used an extended query which specified: integer linear programming (ILP), dynamic programming (DP), greedy approach, evolutionary algorithm (EA) (that encompasses the previously considered class of genetic algorithms) as well as the widely popular nowadays machine learning (ML). Specifically, for each of these formulations, we ran a query as follows: <problem formulation> AND <"resource" OR "resource aware problems"> and sorted the results by relevance. Scopus provides details on how relevance is computed[2] which considers: Number of hits, how significant the word is, position in the document and occurrence in title, keywords etc., proximity of terms and completeness in terms of the words from the query. Finally, out of each of these queries we analyzed top 50 works in terms of problems in specific domains, using the given problem formulation. This has increased the number of sources considered very considerably. Additionally, several new applications/domains have been distinguished, along with new general type resources identified in the works.

3. Resource-aware Problems Across Domains with Resources and Problem Formulations

3.1. Resources, Formulations and Applications/Domains

Within this paper we use the term resource in a broad context that encompasses two classes of assets, that can refer to both physical and non-physical forms:

1. problem specific resources – entities and assets that show up in the context of an optimization problem in a given domain. For instance, in the case of resource allocation in cloud computing, such resources would include: computational nodes with CPUs, GPUs, storage, network, applications.
2. general resources – entities and assets that are of interest in optimization problems in potentially various domains that can exist either in a physical or in a non-physical form. Examples of these include: time, monetary/other cost, energy used, etc. As indicated before, these can in fact be metrics describing the use of particular physical resources e.g. response/execution time of an application run in a computer system at the given cost with a certain amount of energy used within the execution time frame.

In order to classify problems considered in possibly various domains, we have decided to distinguish selected, frequently used problem formulations/approaches used for stating problems formally which can be subsequently solved using specific algorithms. The formulations we distinguish are as follows: integer linear programming (ILP); dynamic programming (DP); greedy approach (GrA); evolutionary algorithms (EA), including genetic

[2] https://service.elsevier.com/app/answers/detail/a_id/14182/supporthub/scopus/

algorithms (GA) considered previously in paper [39] as well as the very popular machine learning (ML).

Furthermore, we aim at assignment of specific optimization problems considered in research works to particular domains, i.e., cloud systems, grid systems, IoT, medical, education, manufacturing etc.

3.2. Classification of Problems in Terms of Resources, Formulations and Domains

Classification of the research works, selected using the methodology outlined in Section 2, was performed separately by problem formulation. Then, we recorded all found problem domains in the given formulation in the respective tables. For each considered paper, we identified a given specific optimization problem and classified it in terms of: resources / metrics used, formulation[3] adopted (possibly more detailed description when applicable) and assignment to a particular domain. Classification of these is included in Tables 1,2,3,4, 5 for ILP, GrA, DP, EA and ML respectively.

Table 1. Selected resource-aware problems by resources / metrics and domain, using ILP formulation

problem description	resources / metrics	formulation	domain	bib
allocating resources for fighting forest fires	human resources; time; financial cost	ILP	wildfire suppression; wildfire simulation	[145]
Mixed-Integer Linear Programming for Resource Constrained Project Scheduling Problem	jobs; projects; time; resources for executing jobs	ILP	general cross domain applicable	[9]
minimization of: electricity cost, CO2 emission, energy import, fossil resource usage, maximization of: employment, social acceptance	solar energy; wind energy; coal energy; natural gas energy; hydroelectric energy; nuclear energy	MOMILP	energy sector	[193]
allocation of health care resources (treatments, population, healthcare programs)	health care resources; financial cost	ILP	healthcare domain; maximization of benefit	[48]
finding the minimum power loss configuration of the network	power distribution network resources	ILP	resource optimization in power distribution networks	[24]
site selection of a wind power plant	energy; power plant	ILP	energy sector	[10]

Continued on next page

[3] for explanation of less frequently appearing abbreviations see Appendix A

Table 1 – continued from previous page

problem description	resources / metrics	formulation	domain	bib
decision-CPM network in order to obtain an overall optimum including time, cost, quality and safety in a road building project	time; cost; quality; safety	ILP	road construction	[150]
scheduling resources in systems that integrate humans with hardware and software components	staff; time; cost; resources assigned by staff	ILP	hospital resource management; simulation	[155]
data assignment optimization in a hybrid heterogeneous environment	computer resources; time	ILP	high performance computing	[21]
cloudlet selection in the multi-cloudlet environment, selection of cloudlet(s), selection of VMs for cloudlets	computing; storage; network	ILP	cloud computing	[102]
Data-center power-aware management, efficient utilization of available resources	data-center resources; power; time	ILP	high performance computing	[58] [154]
scheduling of satellite observations	observation capabilities of satellites; mission time constraints	ILP	satellite Earth observations	[34]
hospital capacity assessment	hospital resources; number of patients; treatment time	MILP	healthcare	[30]
agricultural water management under uncertainty	water resources; ecological water requirements; uncertainty levels	MILP	agriculture; water allocation	[184]
preventive maintenance scheduling	cost; reliability; resources;	MILP	generic preventive maintenance	[111]
mobile workforce scheduling	traveling cost; action cost; teams; task	MILP	mobile workforce scheduling	[192]
Volt/var optimization of unbalanced power distribution networks	transformers; reactive power resources; embedded generators	MILP	power distribution networks	[25]
selection of an appropriate agent in a military confrontation	properties of combat agents; properties of combat forces	MILP	military operations	[15]

Continued on next page

Table 1 – continued from previous page

problem description	resources / metrics	formulation	domain	bib
allocation and sequencing of elective operations on hospital operating rooms	operations; human resources; time; schedule	MILP	healthcare	[107]
continuous berth allocation	quayside resources; vessels; time;	ILP	ship terminal management	[181]
bus scheduling	bus seats demand; bus seats supply;	MILP	public transport scheduling	[116]
optimization of building energy use	electricity sources; electricity cost; grid power import/export schedule	MILP	smart grid; smart home	[71]
carrier optimization in wireless localization networks	network resources; power allocation; spectrum allocation	MILP	wireless networks	[183]
optimization of humanitarian aid resource distribution time	distance; vehicle density; travel time; aid resources demand	MILP	disaster response	[2]
telescope network scheduling	astronomers; reservations; preferences	ILP	astronomy	[93]
planning and operations of renewable energy-based distributed power systems	energy cost; energy supply availability; energy sources; optimal energy source sizes	ILP	smart grid; renewable energy	[41]
optimization of multi-period investment planning in street lighting systems	energy savings; budget constraints; state of the system; available technologies	MILP	streetlight systems; investment planning	[144]
optimal selection and sizing of a smart building system	thermal storages; electrical storages; heating and cooling systems; renewable energy sources; policies; cost	MILP	low-energy building design	[11]
dynamic optimal nurse scheduling	nurses; tasks; constraints; locations; preferences; work regulations	ILP	healthcare	[72]

Table 2. Selected resource-aware problems by resources / metrics and domain, using greedy formulation

problem description	resources / metrics	formulation	domain	bib
dynamic multi-user resource allocation in the downlink of OFDMA system, power consumption minimization	communication channels; power consumption	GrA	resource allocation; telecomm.	[121]
scheduling of flows from various applications in overload states, downlink scheduling	throughput; loss; time (delay)	GrkA	resource allocation; telecomm.	[53]
preparation of educational schedule in the higher education	human resources; classes; courses; time; cost	GrA	education	[133]
allocating resources in Virtual Sensor Networks, maximizing revenue of multiple concurrent applications' schedule	processing power; bandwidth; storage; time; energy	GrA	Virtual Sensor Networks	[27]
Set Covering Problem as a template for resource management	generic resources; time	wGrA	resource management	[156]
Maximizing utility and revenue of hardware resources in virtual machine allocation	processing power; memory; storage	GrA	datacenter provisioning	[136] [137]
Reducing task duplication in task scheduling on heterogeneous distributed systems	computational resources	GrA	distributed computing	[1]
Task offloading and resource allocation in power network monitoring (PIoT)	computational resources; communication resources	GrA	power network monitoring	[98]
Resource-aware fluid scheduling	computational resources; communication resources; fluids	GrA	physics modeling	[182]
task scheduling in a cloud computing environment, with time and energy constraints	energy consumption; time	GrA	cloud computing	[165]
radio resource allocation and interference management	link performance; cell throughput	GrA	telecomm.	[161]

Continued on next page

Table 2 – continued from previous page

problem description	resources	formulation	domain	bib
allocation of resources for data traffic in 5G networks	network resources; quality of service; resource scheduling	GrA	telecomm.	[47]
allocation of resources for online teaching	course resources; network; bandwidth; delay	GrA	online education	[173]
dynamic battlefield resource scheduling	campaign resources	GrA	military	[160]
combinatorial auctions in efficient cloud resource allocation	cloud resources; resource pricing	aGrA	cloud computing	[35]
computing resource scheduling in the computing-aware network	computing resources; QoS attributes; network; tasks	dGrA	edge computing; IoT; internet-of-vehicles	[95]
allocation or constrained resources to multi-activity projects	human resources; equipment; materials;	GrA	manufacturing industry	[100]
HW/SW partitioning in SoC design	task criticality; time savings; task frequency; task area	GrA	System-on-Chip design	[167]
relief resource allocation to areas of disaster	equity constraint; relief resource demand; relief resources	GrA	relief operations	[61]

Table 3. Selected resource-aware problems by resources / metrics and domain, using dynamic programming formulation

problem description	resources / metrics	formulation	domain	bib
agriculture and natural resources management	natural resources	DP	agriculture; natural resources	[86]
scheduling water resources; minimization of cost of running a hydroelectric system	water resources; cost	DP	power systems	[32]
stochastic resource allocation	generic resources; financial cost; time	DP	general resource allocation	[56]
stochastic resource allocation	ships; weapons; time; security	DP	military real-time naval operations	[130]

Continued on next page

Table 3 – continued from previous page

problem description	resources / metrics	formulation	domain	bib
HPC compute nodes allocation	application specific resources; accelerators; storage	DP	HPC	[29]
dynamic code loading	grid resources; power consumption	DP	dynamic reconfiguration of servers	[119]
Balancing resources in robotic vision	computational power; bandwidth; responsiveness	DP	balanced utilization of computing resources	[125]
integration of low cost wearable sensors, processing of sensors' data at the cloud edge	energy; bandwidth; processing power; measurement quality	DP	healthcare; clinical-level continuous patient monitoring	[6]
Seamless image manipulation	still images	DP	image processing	[12]
task scheduling and resource allocation in distributed systems	computing resources; cost	DP	distributed processing	[63] [142] [131]
planning water resources management systems under uncertainty	water resources	DIRSDP	water resources management	[105]
hydraulics and water resources simulating, optimizing water transfer system	water resources	DP	agriculture; water consumption	[110]
stochastic dynamic programming for military applications	military resources; financial cost	DP	determining soldiers/ medical support location	[80]
data center resource dynamic scheduling for energy optimization, emission reduction	energy; time; computational resources; physical resources	DP	data center optimization	[97]
finding the optimal bidding strategy for a firm	resources available to the firm	infinite horizon semi-Markov DP	public tenders in oligopolistic market	[70]
bandwidth allocation in OFDM systems with rate constraint to minimize transmission power	bandwidth; user profiles	aDP	telecomm.	[75]

Continued on next page

Table 3 – continued from previous page

problem description	resources / metrics	formulation	domain	bib
sensor resource management	time to acquire target; target priorities; sensor field of view	sDP	surveilance (civil and military)	[171]
optimization of energy purchase and production	energy sources	DP	energy market	[109]
dynamic fleet management	vehicles; vehicle states; customer demands	aDP	vehicle fleet management	[64]
optimization of resource allocation in a factory	production line resources; profit	DP	industry	[172]
price management, maximizing revenue	customer; resource (requests)	aDP & sDP	price management systems	[57]
optimization of water treatment and allocation	water resource; resource state	DP	environmental resources allocation	[187]
resource allocation in R&D projects	project; activities; cost;	DP	cost optimization in R&D projects	[87]
resource allocation to cloud storage	storage; efficiency; load	aDP	cloud computing	[141]
operation of a water reservoir system	water reservoirs; reservoir state; operation policy	DP	water resource planning	[17]
resource-constrained project scheduling	resources; resource availability	aDP with Markov decision process	applicable to many fields	[175]
resource allocation in industrial maintenance	human resources; equipment; time	DP	heavy industry	[62]
finding optimal preventive maintenance budget in power distribution network with reliability constraints	maintenance resources; reliability constraints	DP	power distribution networks	[14]
resource allocation in sliced 5G radio access networks	rate; latency; reliability; separation	DP with hierarchical auction	telecomm.	[153]
assembly line balancing	resource constraints; task precedence relations	DP	manufacturing	[135]
optimization of regional industrial structure development	labor; capital; energy; natural resources; technological progress	grey DP	economy	[126]

Continued on next page

Table 3 – continued from previous page

problem description	resources / metrics	formulation	domain	bib
reducing stochastic errors in accelerometers and gyroscopic sensors	computational resources	DP	metrology	[113]

Table 4. Selected resource-aware problems by resources / metrics and domain, using evolutionary algorithms

problem description	resources / metrics	formulation	domain	bib
resource provisioning and scheduling in uncertain cloud environments	financial cost; time; deadlines imposed	GA	cloud computing	[31]
resource-constrained project scheduling with transfer times	generic resources; transfer time	GA	cross domain applicable problem formulation	[82]
resource constrained multi-project scheduling	generic resources; time	GA	cross domain applicable problem formulation	[66]
resource constrained project scheduling - comparison of GAs	generic resources; time	multiple GA	cross domain applicable problem formulation	[60] [5] [101]
		GA parameter tuning		[162]
		decomposition based GA		[43]
		quantum inspired GA		[149]
		Elitist GA		[94]
construction scheduling	generic resources; bridge; time	GA	general problem formulation; bridge construction	[163]
troops-to-tasks problem	military resources; time	GA	military field applications	[52,51]
grid resource allocation	grid resources; time	GA	grid computing	[49]
regional drinking water supply	water resources; financial cost; ecological value; energy	GA	water resource research	[166]
groundwater management	water resources; financial cost; environmental value; time	GA	water resource research	[88]
surgery scheduling	hospital resources; time	GA	healthcare sector	[143]

Continued on next page

Table 4 – continued from previous page

problem description	resources / metrics	formulation	domain	bib
scheduling problems on flexible manufacturing systems (FMS)	machines; storage buffers; material; tool-changing devices; fixtures; pallets; time	GA+PSO	manufacturing system	[54]
protection of marine environment and allocation of response vessels to minimize costs of oil spill at sea	cost; time; environmental burden	GA	environmental protection	[194]
Power aware resource reconfiguration	resources; power consumption	GA	cloud computing	[44]
processing of time-constrained workflows in mobile edge computing	resources; power limitations	GA	mobile edge computing	[83]
power-aware allocation of virtual machines in a cloud	energy; power consumption	GA	cloud computing; virtualization	[134]
Solving resource constraints in fog computing	fog computing resources	GA	Fog-cloud computing; Internet of Things	[74]
virtual network embedding onto underlying physical infrastructure	physical infrastructure; network topology	GA	network virtualization	[190]
scheduling in grid resource management	grid resources; cost; time	EA + learning	grid computing	[159]
design of combinational logic circuits	circuit; gate; cost; time	EA	electronics	[185]
dynamic multicast routing with network coding	network topology; cost; time	EA	telecomm.	[176]
multi-agent coalition formation	agents; tasks; cost; time	IMOEA	multi-agent processing	[177]
employment level planning for assigned construction project lead time	human resources; project; time	GA+HEA	project management	[146]
optimization of subcarrier allocation and transmit power	network; time	EA	telecomm.	[99]
multi-period dynamic emergency resource scheduling	roads; time	MOEA/D-mdERS	post-disaster emergency resource scheduling	[189]
resource planning and scheduling of payload	space resources	PEA	space (satellite)	[96]

Continued on next page

Table 4 – continued from previous page

problem description	resources / metrics	formulation	domain	bib
order quantities in a multi-item inventory with constraints on storage space and capital	storage; cost	two-phase EA	retail	[81]

Table 5. Selected resource-aware problems by resources / metrics and domain, using machine learning formulation

problem description	resources / metrics	formulation	domain	bib
resource allocation, optimization of the downlink communication [76], resource allocation for 5G [140], medium access control in 6G [89]	network resources	sML, RL	wireless systems; telecomm.	[76] [140] [89]
fog computing resource management review	cost; energy; throughput; time; task	NN, RL, DT, etc.	fog computing	[50]
resource planning system for grocery retail delivery services	groceries; customer; driver; cost	ML	grocery retail	[178]
highlighting geologic sweet spots for multiple US onshore basins	natural resources	ML	geology	[28]
ML for tourism information system, optimization of economy of scenic spots	cost; tourism resources	GBDT, Lambdamart	tourism; economy	[191]
using ML for hydrological modeling, flood forecasting, drought prediction, water resource management	water resources; cost; time	ANNs, RMTs, DL, RNNs, LSTM	water resources management	[128]
compression of quantum data	information	ML	quantum computing	[127]
identification of groundwater potential zones	water resources	EBM, GAMI-net	water resource research	[40]
pronominal coreference resolution using machine learning	text corpus	KNN, LR, XGBoost	languge research	[16]
machine learning-based handoff management in 5G networks	energy; network topology; resource allocation	ML	wireless networks; telecomm.	[139]

Continued on next page

Table 5 – continued from previous page

problem description	resources / metrics	formulation	domain	bib
interpretable machine learning methods and their applications in the field of information resource management	information resources	RMs, DTs, attention mechanisms, PDP, ICE, PFI, LIME, SHAP	public opinion research; social network user behavior; healthcare; scientometric research	[106]
soil moisture prediction through machine learning	natural resources	ML	environmental; water resources management	[85]
ML based employee engagement, appraisal, organizational culture prediction [147], recruitment procedures[77]	human resources	DTs, LR[147], sML[77]	human resource systems management	[147] [77]
mineral resource estimation, exploration	natural resources	SVM, SVR, and ANN used for MRE, mostly RF, neuro-fuzzy, SVM, and ANN ML	management of natural resources	[108] [46] [26]
multi-core resource management	computer resources	RL, ANNs	computer resource management	[112]
water quality prediction	water resources; time	DNNs	water research	[157] [104]
workload prediction in serverless environments	computer resources; cost	LSTM, ARIMA, VAR	serverless computing	[123]
sharing digital education training resources[179], personalized learning[168]	information; training resources; students	SVMs, DT, NNs	education	[179] [168]
increasing the resource efficiency of screw-fastening process	screws; cost	DT, SVM, ANNs	manufacturing	[114]
predicting confirmed cases and trend, classification and diagnosis, medical management	medical resources	ML	medical	[7]

Continued on next page

Table 5 – continued from previous page

problem description	resources / metrics	formulation	domain	bib
resource provisions, scheduling, allocation, energy efficiency, resource[164] management[67] resource scheduling[103]	cloud resources; time; cost; energy	regression, NNs, DTs, RL, SVM	cloud resource management	[164] [67] [103]
resource-efficient computation offloading in IoT devices	computer resources; time	ML + backward induction	IoT; edge computing; cloud computing	[19]
project resource allocation	project resources; cost; time	SVM	project management	[148]
water availability prediction	natural resources; natural phenomena	NNs, LSTM, SVM, etc.	water research	[104]
intrusion detection system for IoT	computer resources; time; memory	logistic regression, passive-aggressive classifiers; perception	IoT	[42]
vehicular network resource allocation strategy	vehicles; network; cost; time	DL, RL, regression	vehicular distributed system	[124]

Additionally, during research we have encountered works that consider various and mixed formulations. Selected examples of these are shown in Table 6, described in terms of the same features as works in the previous tables.

Table 6. Selected resource-aware problems by resources / metrics, mixed formulations

problem description	resources / metrics	formulation	domain	bib
scheduling service based workflow applications with changeable service availability	time; cost	ILP, GA, GAIN, divide-and-conquer	scientific workflows; business workflows; mixed workflows	[38]
performance and energy trade-off analysis for running parallel applications on heterogeneous multi processing systems	execution time; energy	(Halton number) sampling of configuration space for Pareto front generation	HPC	[36]

Continued on next page

Table 6 – continued from previous page

problem description	resources / metrics	formulation	domain	bib
performance-energy optimization for parallel applications using power capping, for CPUs and GPUs	time; energy	linear configuration space exploration	HPC	[91,90,92]
tugboat allocation optimization in container terminals	vessels; tugboats; time	combined GA + ant colony optimization	marine research	[169]
approximate DP for resource management in multi-cloud environments	cloud resources; time; revenue	approximate DP, RL	cloud resource management	[129]
allocation method of wind resources under the background of carbon neutralization	natural resources; energy; cost	EA, LP	wind energy; natural resource management	[188]
comb jamming resource allocation algorithm	data/information	greedy + EA	telecomm.	[174]
optimal financial investment of limited resources in enterprise	risk; benefit; time; financial resources	DP and GA	investment management; financial	[65]
virtual network function (VNF) scheduling and deployment	resource cost; delay-satisfied request ratio	ILP + greedy	software-defined networks; telecomm.	[186]
optimal multi-resource allocation in big data mining model training	resources; tasks; parallelism; resource constraints	greedy + GA	big data model training	[170]

We shall note that performing the extended search for the articles from the Scopus database, we generally identified different articles than those in the original paper [39]. There was almost no overlap between current and previous search results. On the other hand, though, the set of domains of identified problems in the two searches mostly matched.

4. Summary of Problem Formulations, Resources and Domains

Based on the classification of the research works shown in the previous section, we can now perform comprehensive analysis concerning:

1. which resources are used in particular problem formulations referring to practical applications,
2. which problem formulations are typically used in particular applications and domains,

3. which resources typically occur in the context of a given application and domain which in fact denotes which of these are considered in the process of an optimization problem in a given domain.

Such analysis allows us to draw conclusions regarding whether:

1. a particular problem formulation is used in the majority of domains,
2. there are formulations that are specific for particular applications/domains,
3. there is a resource that is used only with a specific problem formulation.

It should be noted that this analysis was performed for the source data used within this paper and outlined in Tables 1 through 6. This does mean that the following results reflect the source data analyzed in the paper rather than the whole set of existing research works.

We shall note during preparation of the following summary results we considered the most frequently occurring resources, without problem-specific ones, as well as applications. Integration of the results from the aforementioned tables required relevant generalization of terms used by respective authors in specific problem formulations. Furthermore, in the following Tables 7 and 8, we counted occurrences of terms corresponding to resources and domains per article i.e. possibly several energy-related terms in an article shown before would be counted as one reference to energy. In Table 9 we placed counts of relevant tuples of a resource and a domain and there can be several such tuples resulting from one article.

Resources considered with various problem formulations are shown in Table 7.

Table 7. Resources identified in various problem formulations, notation: I/M – I denotes the number of occurrences in individual formulations, M – denotes the number of occurrences in mixed formulations

resource	**ILP**	**GrA**	**DP**	**EA**	**ML**	**sum**
time	11/2	7/3	5/2	22/2	8/1	63
monetary resources	10/3	1/2	6/1	9/2	9/	43
energy	13/1	3/	5/	4/1	3/	30
human resources	10/	2/	2/	1/	4/	19
computer, network, storage	8/	17/	11/1	6/	8/1	52
natural resources	5/1	/	8/	2/1	7/	24

Continued on next page

Table 7 – continued from previous page

resource	ILP	GrA	DP	EA	ML	sum
resources in general problem formulations	6/	6/	6/	8/	/	26
data/information	/	/1	1/	/1	4/	7
sum	70	42	48	59	45	264

Applications that are considered in various problem formulations are presented in Table 8.

Table 8. Applications for which selected problem formulations are used, notation: I/M – I denotes the number of occurrences in individual formulations, M – denotes the number of occurrences in mixed formulations

application	ILP	GrA	DP	EA	ML	sum
power/energy	6/	1/	3/	/	/	10
general resource management	4/1	3/	4/	10/1	/	23
computer resource management	3/1	8/1	6/1	10/2	9/1	42
communication	1/1	5/2	2/	3/1	4/	19
education	/	2/	/	/	1/	3
natural resources management	3/1	/	8/	3/1	8/	24
military applications	1/	1/	3/	1/	/	6
retail	/	/	2/	1/	2/	5
tourism	1/	/	1/	/	1/	3
manufacturing	/	1/	4/	2/	1/	8
medical/health	5/	/	4/	1/	3/	13

Continued on next page

Table 8 – continued from previous page

application	**ILP**	**GrA**	**DP**	**EA**	**ML**	**sum**
human resources management	2/	1/	/	/	1/	4
transport	3/	/	1/	1/1	/	6
space	2/	/	/	1/	/	3
disaster management	1/	1/	/	1/	/	3
sum	36	26	39	40	31	172

Additionally, we identify how resources are considered within selected applications/domains. Such assessment, based on the reviewed papers, is included in Table 9.

Table 9. Resources identified in selected applications/domains

resource	**power/energy**	**general res mgmt**	**computer res mgmt**	**communication**	**education**	**nat res mgmt**	**military**	**retail**	**tourism**	**manufacturing**	**medical/health**	**human res mgmt**	**transport**	**space**	**disaster management**	**sum**
time	/	10/1	16/3	6/1	2/	4/	2/	1/	/	5/	4/	2/	4/1	1/	3/	66
monetary resources	4/	4/1	8/1	2/1	1/	5/1	1/	2/	1/	3/	2/	3/	3/	1/	3/	47
energy	9/	/	12/2	6/	/	4/1	/	/	/	/	1/	/	1/	/	/	36
human resources	/	1/	/	1/	2/	/	2/	2/	1/	2/	6/	4/	2/	1/	4/	28
computer, network, storage	7/	/	32/1	14/	2/	/	1/	/	/	/	2/	/	1/	/	/	60
natural resources	11/	/	/	/	/	22/	/	/	/	/	/	/	/	/	/	33
resources in general problem formulations	/	14/	2/	1/	/	1/	/	/	/	1/	2/	/	2/	/	/	23

Continued on next page

Table 9 – continued from previous page

resource	power/energy	general res mgmt	computer res mgmt	communication	education	nat res mgmt	military	retail	tourism	manufacturing	medical/health	human res mgmt	transport	space	disaster management	sum
data/information	/	2/	/	1/	3/	/	/	/	/	/	/	1/	/	/	/	7
sum	31	33	77	33	10	38	6	5	2	11	17	10	14	3	10	300

Based on this analysis we can draw the following conclusions:

1. All the problem formulations are similarly frequent across applications (total), as can be seen from Table 8. The same can be seen across the resources used, as shown in Table 7.
2. Not surprisingly, as shown in Tables 7 and 9, time and cost are the most frequently addressed non-physical resources, followed by energy. Out of the physical resources, computer, network and storage devices are most frequently considered. Across applications/domains, computer system management, natural resource management, general universally applicable resource management problems, and communication are the most frequently considered ones.
3. ML targets all but general resources and appears in most of the specific contexts, as it is linked to particular applications. This also emphasizes its popularity nowadays.
4. While data/information as a resource is present during optimization using GrA+EA, DP and ML, it is not as frequently considered as the other resources like time, energy, cost.
5. From Table 8 we can see that within the set of papers analyzed, papers on tourism tend to use ILP, DP and ML approaches rather than GrA and EA. Retail domain seems to omit ILP and GrA formulations. While we know that ML can be used for disaster management e.g. in [33,78], this has not been visible in our set of papers, suggesting it is an area worthy of further exploration. The same would apply to military and space domains.
6. From Table 9 we can see that time and cost are practically considered in all identified fields, there is room for further energy-aware research in many fields, including: education, retail, tourism, manufacturing and transport. While, in some of these, energy aspects can be considered within costs, energy considerations, especially concerning environmental impact, are becoming more and more important and are likely to require more direct exposure. Other interesting cross resource domain combinations that could be further explored, in our opinion, include: more focus on human resources in the computer resources management, as well as more focus on consideration of natural resources in contexts other than those specifically focused on natural resource management, as visible in Table 9. Finally, data/information per se is

not deeply present as a resource in other domain-specific areas, other than in works specifically focused on general resource management models and algorithms, education, communication and social contexts.

5. Summary and Future Work

We were able to identify resources and metrics used in various problem formulations as well as problem formulations typically used in a given application/domain. Additionally, we mapped particular resources to applications/domains which allows to draw conclusions about their perceived importance.

Resource identification in Table 9 shows that time and monetary resources are always considered as important, while energy is explicitly considered in 1/3rd of domains and natural resources are given even less direct consideration. It would be interesting to conduct a similar literature survey in, e.g., five years and check, whether increased awareness of energy cost and of demand pressure on natural resources will be reflected in the repeated survey findings. Furthermore, the search for source research works could be extended to include other scientific (indexing) databases, including: ACM DL, IEEE Xplore, Web of Science etc.

Ongoing research in this field has a potential for new formulations. Such occurrences could trigger a new research to amend our findings.

Acknowledgments. This work is partially supported by CERCIRAS COST Action CA19135 funded by COST.

A. Abbreviations

aDP – approximate Dynamic Programming;
aGrA – adaptive Greedy Algorithm;
ANN – Artificial Neural Network;
ARIMA – Auto Regressive Integrated Moving Average;
dGrA – dynamic Greedy Algorithm;
DIRSDP – Dual Interval Robust Stochastic Dynamic Programming;
DNN – Deep Neural Network;
DT – Decision Trees;
EBM – Explainable Boosting Machine;
GBDT – Gradient Boosting Decision Trees;
GrkA – Greedy knapsack Algorithm;
HEA – Hybrid Evolutionary Algorithm;
ICE – Individual Conditional Expectation;
IMOEA – Improved Multi-Objective Evolutionary Algorithm;
KNN – k-nearest neighbors;
LIME – Local Interpretable Model-agnostic Explanations;
LP – Linear Programming;
LR – Logistic Regression;
LSTM – Long Short-Term Memory;

MILP – Mixed Integer Linear Programming;
MOEA/D-mdERS – Multi-Objective Evolutionary Algorithm for Dynamic multi-period dynamic Emergency Resource Scheduling;
MOMILP – Multi Objective MILP;
MRE – Most Relevant Explanation;
NN – Neural Network;
PDP – Partial Dependence Plot;
PEA – Plasmodium Evolutionary Algorithm;
PFI – Permutation Feature Importance;
PSO – Particle Swarm Optimization;
RF – Random Forest;
RL – Reinforcement Learning;
RMT – Regression and Model Trees;
RNN – Recurrent Neural Network;
sDP – stochastic Dynamic Programming;
SHAP – SHapley Additive exPlanations;
sML – supervised Machine Learning;
SVM – Support Vector Machine;
SVR – Super Vector Regression;
VAR – Vector Auto Regression;
wGrA – weighted Greedy Algorithm.

References

1. A resource-aware scheduling algorithm with reduced task duplication on heterogeneous computing systems. J Supercomput 68, 1347—1377 (2014)
2. Acero Condor, A.A., Ramirez Castañeda, C.M., Taquía Gutiérrez, J.A.: Optimization of humanitarian aid resource distribution time through mixed integer linear programming. p. 191 – 197 (2023)
3. Akter, S., Wamba, S.F.: Big data and disaster management: a systematic review and agenda for future research. Annals of Operations Research 283, 939 – 959 (2017)
4. Al-Ahmad, A.S., Kahtan, H.: Cloud computing review: Features and issues. In: 2018 International Conference on Smart Computing and Electronic Enterprise (ICSCEE). pp. 1–5 (2018)
5. Alcaraz, J., Maroto, C.: A Robust Genetic Algorithm for Resource Allocation in Project Scheduling. Annals of Operations Research 102(1), 83–109 (February 2001)
6. Amiri, D., Anzanpour, A., Azimi, I., Levorato, M., Liljeberg, P., Dutt, N., Rahmani, A.M.: Context-aware sensing via dynamic programming for edge-assisted wearable systems 1 (03 2020)
7. An, Q., Gao, Q., Gao, Z., Qian, Y.: A survey of machine learning technologies for covid-19 pandemic. p. 7 – 11. Institute of Electrical and Electronics Engineers Inc. (2022)
8. Ang, T.Z., Salem, M., Kamarol, M., Das, H.S., Nazari, M.A., Prabaharan, N.: A comprehensive study of renewable energy sources: Classifications, challenges and suggestions. Energy Strategy Reviews 43, 100939 (2022)
9. Araujo, J.A.S.: Mixed-Integer Linear Programming Based Approaches for the Resource Constrained Project Scheduling Problem. Ph.D. thesis, Universidade Federal de Ouro Preto (December 2019)
10. Ari, E.S., Gencer, C.: Proposal of a novel mixed integer linear programming model for site selection of a wind power plant based on power maximization with use of mixed type wind turbines. Energy & Environment 31(5), 825–841 (2020)

11. Ashouri, A., Fux, S.S., Benz, M.J., Guzzella, L.: Optimal design and operation of building services using mixed-integer linear programming techniques. Energy 59, 365 – 376 (2013)
12. Avidan, S., Shamir, A.: Seam carving for content-aware image resizing. In: ACM Trans. Graph. p. 10. SIGGRAPH (2007)
13. Avoine, G., Junod, P., Oechslin, P.: Characterization and improvement of time-memory trade-off based on perfect tables. ACM Trans. Inf. Syst. Secur. 11(4) (Jul 2008)
14. Bacalhau, E.T., Usberti, F.L., Lyra, C.: A dynamic programming approach for optimal allocation of maintenance resources on power distribution networks (2013)
15. Baek, S., Moon, S., Kim, H.J.: Entry optimization using mixed integer linear programming. International Journal of Control, Automation and Systems 14(1), 282 – 290 (2016)
16. Barbu, E., Muischnek, K., Freienthal, L.: A study in estonian pronominal coreference resolution. Frontiers in Artificial Intelligence and Applications 328, 3 – 10 (2020)
17. Bayazit, M., Duranyildiz, I.: An iterative method to optimize the operation of reservoir systems. Water Resources Management 1(4), 255 – 266 (1987)
18. Bermudo Mera, J.M., Karmakar, A., Verbauwhede, I.: Time-memory trade-off in toom-cook multiplication: an application to module-lattice based cryptography. IACR Transactions on Cryptographic Hardware and Embedded Systems 2020(2), 222–244 (Mar 2020)
19. Bharti, P., Chaudhary, S., Snigdh, I.: A novel machine learning approach to delay efficient offloading strategy for mobile edge computing. Lecture Notes in Electrical Engineering 887, 215 – 221 (2023)
20. Bly, J., Francescutti, L.H., Weiss, D.: Disaster management: A state-of-the-art review. In: Farsangi, E.N. (ed.) Natural Hazards, chap. 1. IntechOpen, Rijeka (2020)
21. Boiński, T., Czarnul, P.: Optimization of Data Assignment for Parallel Processing in a Hybrid Heterogeneous Environment Using Integer Linear Programming. The Computer Journal (02 2021)
22. Bonnin, M., Azzaro-Pantel, C., Domenech, S.: Optimization of natural resource management: Application to french copper cycle. Journal of Cleaner Production 223, 252–269 (2019)
23. Boon, C., Hartog, D.N.D., Lepak, D.P.: A systematic review of human resource management systems and their measurement. Journal of Management 45(6), 2498–2537 (2019)
24. Borghetti, A.: Mixed Integer Linear Programming Models for Network Reconfiguration and Resource Optimization in Power Distribution Networks, chap. 2, pp. 43–88. John Wiley & Sons, Ltd
25. Borghetti, A., Napolitano, F., Nucci, C.A.: Volt/var optimization of unbalanced distribution feeders via mixed integer linear programming (2014)
26. Bournas, N., Touré, A., Balboné, M., Zagré, P.S., Ouédraogo, A., Khaled, K., Prikhodko, A., Legault, J.: Use of machine learning techniques on airborne geophysical data for mineral resources exploration in burkina faso. Exploration Geophysics 2019(1), 1 – 4 (2019)
27. Bousnina, S., Cesana, M., Ortín, J., Delgado, C., Gállego, J.R., Canales, M.: A greedy approach for resource allocation in virtual sensor networks. In: 2017 Wireless Days. pp. 15–20 (2017)
28. Bowman, J., Tabatabaie, H., Bowman, J.A.: A comparative study of machine learning model results and key geologic parameters for unconventional resource plays. p. 470 – 486. Unconventional Resources Technology Conference (URTEC) (2021)
29. Braun, M., Buchwald, S., Mohr, M., Zwinkau, A.: Dynamic x10: Resource-aware programming for higher efficiency. Tech. Rep. 8, Karlsruhe Institute of Technology (2014)
30. Burdett, R.L., Kozan, E., Sinnott, M., Cook, D., Tian, Y.C.: A mixed integer linear programing approach to perform hospital capacity assessments. Expert Systems with Applications 77, 170 – 188 (2017)
31. Calzarossa, M.C., Massari, L., Nebbione, G., Della Vedova, M.L., Tessera, D.: Tuning genetic algorithms for resource provisioning and scheduling in uncertain cloud environments: Challenges and findings. In: 2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP). pp. 174–180 (2019)

32. Castellano, A., Martínez, C., Monzón, P., Bazerque, J.A., Ferragut, A., Paganini, F.: Quadratic approximate dynamic programming for scheduling water resources: a case study (2020)
33. Chamola, V., Hassija, V., Gupta, S., Goyal, A., Guizani, M., Sikdar, B.: Disaster and pandemic management using machine learning: A survey. IEEE Internet of Things Journal 8(21), 16047–16071 (2021)
34. Chen, X., Reinelt, G., Dai, G., Spitz, A.: A mixed integer linear programming model for multi-satellite scheduling (2018)
35. Chichin, S., Vo, Q.B., Kowalczyk, R.: Adaptive market mechanism for efficient cloud services trading. p. 705 – 712 (2014)
36. Coutinho Demetrios, A.M., De Sensi, D., Lorenzon, A.F., Georgiou, K., Nunez-Yanez, J., Eder, K., Xavier-de Souza, S.: Performance and energy trade-offs for parallel applications on heterogeneous multi-processing systems. Energies 13(9) (2020)
37. Cramer, E.J., Dennis, Jr., J.E., Frank, P.D., Lewis, R.M., Shubin, G.R.: Problem formulation for multidisciplinary optimization. SIAM Journal on Optimization 4(4), 754–776 (1994)
38. Czarnul, P.: Comparison of selected algorithms for scheduling workflow applications with dynamically changing service availability. J. Zhejiang Univ. Sci. C 15(6), 401–422 (2014)
39. Czarnul, P., Matuszek, M.: Identification of selected resource-aware problems across scientific disciplines and applications. In: Iyenghar, P., Rakić, G. (eds.) Proceedings of the First Workshop on Connecting Education and Research Communities for an Innovative Resource Aware Society (CERCIRAS). CEUR, vol. 3145. Novi Sad, Serbia (September 2021)
40. Dahal, K., Sharma, S., Shakya, A., Talchabhadel, R., Adhikari, S., Pokharel, A., Sheng, Z., Pradhan, A.M.S., Kumar, S.: Identification of groundwater potential zones in data-scarce mountainous region using explainable machine learning. Journal of Hydrology 627 (2023)
41. Dahiru, A.T., Tan, C.W., Salisu, S., Lau, K.Y.: Multi-configurational sizing and analysis in a nanogrid using nested integer linear programming. Journal of Cleaner Production 323 (2021)
42. Data, M., Bakhtiar, F.A.: Resource efficient intrusion detection systems for internet of things using online machine-learning models. p. 297 – 303. Association for Computing Machinery (2023)
43. Debels, D., Vanhoucke, M.: A decomposition-based genetic algorithm for the resource-constrained project-scheduling problem. Operations Research 55(3), 457–469 (2007)
44. Deng, L., Li, Y., Yao, L., Jin, Y., Gu, J.: Power-aware resource reconfiguration using genetic algorithm in cloud computing. Mob. Inf. Syst. 2016, 4859862:1–4859862:9 (2016)
45. Diouani, S., Medromi, H.: Trade-off between performance and energy management in autonomic and green data centers. In: Proceedings of the 2nd International Conference on Networking, Information Systems & Security. NISS19, Association for Computing Machinery, New York, NY, USA (2019)
46. Dumakor-Dupey, N.K., Arya, S.: Machine learning—a review of applications in mineral resource estimation. Energies 14(14) (2021)
47. Elamaran, E., Sudhakar, B.: Greedy based round robin scheduling solution for data traffic management in 5g. p. 773 – 779 (2019)
48. Epstein, D., Chalabi, Z., Claxton, K., Sculpher, M.: Mathematical programming for the optimal allocation of health care resources (2005)
49. Ezugwu, A.E., Okoroafor, N.A., Buhari, S.M., Frincu, M.E., Junaidu, S.B.: Grid resource allocation with genetic algorithm using population based on multisets:. Journal of Intelligent Systems 26(1), 169–184 (2017)
50. Fahimullah, M., Ahvar, S., Agarwal, M., Trocan, M.: Machine learning-based solutions for resource management in fog computing. Multimedia Tools and Applications 83(8), 23019 – 23045 (2024)
51. Fauske, M.F.: Optimizing the troops-to-tasks problem in military operations planning. Military Operations Research 20(4), 49–57 (2015)
52. Fauske, M.F.: Using a genetic algorithm to solve the troops-to-tasks problem in military operations planning. The Journal of Defense Modeling and Simulation 14(4), 439–446 (2017)

53. Ferdosian, N., Othman, M., Ali, B.M., Lun, K.Y.: Greedy–knapsack algorithm for optimal downlink resource allocation in lte networks. Wireless Networks 22(5), 1427–1440 (Aug 2015)
54. Filho, M.G., Barco, C.F., Neto, R.F.T.: Using genetic algorithms to solve scheduling problems on flexible manufacturing systems (fms): a literature survey, classification and analysis. Flexible Services and Manufacturing Journal 26, 408–431 (2014)
55. Foris, D., Popescu, M., Foris, T.: A comprehensive review of the quality approach in tourism. In: Butowski, L. (ed.) Mobilities, Tourism and Travel Behavior, chap. 10. IntechOpen, Rijeka (2017)
56. Forootani, A., Iervolino, R., Tipaldi, M., Neilson, J.: Approximate dynamic programming for stochastic resource allocation problems. IEEE/CAA Journal of Automatica Sinica 7(4), 975–990 (2020)
57. Forootani, A., Liuzza, D., Tipaldi, M., Glielmo, L.: Allocating resources via price management systems: a dynamic programming-based approach. International Journal of Control 94(8), 2123 – 2143 (2021)
58. García, J.L.B., Mestre, R.G., Viñals, J.T.: An integer linear programming representation for data-center power-aware management (2010)
59. Garengo, P., Sardi, A., Nudurupati, S.S.: Human resource management (hrm) in the performance measurement and management (pmm) domain: a bibliometric review. International Journal of Productivity and Performance Management ahead-of-print (2021)
60. Gargiulo, F., Quagliarella, D.: Genetic algorithms for the resource constrained project scheduling problem. In: 2012 IEEE 13th International Symposium on Computational Intelligence and Informatics (CINTI). pp. 39–47 (2012)
61. Ge, H., Liu, N.: An relief resource allocation model with equity constraints. p. 506 – 509 (2011)
62. Ghaeli, M.: A dynamic programming approach for resource allocation in oil and gas industry. Journal of Project Management (Canada) 4(3), 213 – 216 (2019)
63. Gianni M., R., Soon-Wook, H.: Cost-aware dynamic resource allocation in distributed computing infrastructures. International Journal of Contents 2(2) (Jun 2011)
64. Godfrey, G.A., Powell, W.B.: An adaptive dynamic programming algorithm for dynamic fleet management, i: Single period travel times. Transportation Science 36(1), 21 – 39 (2002)
65. Gong, J.Q., Qin, X.H.: The dynamic programming model for investment decision of enterprise. Applied Mechanics and Materials 519-520, 1466 – 1469 (2014)
66. Gonçalves, J., Mendes, J., Resende, M.: A genetic algorithm for the resource constrained multi-project scheduling problem. European Journal of Operational Research 189(3), 1171–1190 (2008)
67. Goodarzy, S., Nazari, M., Han, R., Keller, E., Rozner, E.: Resource management in cloud computing using machine learning: A survey. p. 811 – 816. Institute of Electrical and Electronics Engineers Inc. (2020)
68. Gourisaria, M.K., Samanta, A., Saha, A., Patra, S.S., Khilar, P.M.: An extensive review on cloud computing. In: Raju, K.S., Senkerik, R., Lanka, S.P., Rajagopal, V. (eds.) Data Engineering and Communication Technology. pp. 53–78. Springer Nature Singapore, Singapore (2020)
69. Guest, D.E.: Human resource management and performance: a review and research agenda. The International Journal of Human Resource Management 8(3), 263–276 (1997)
70. Gunter, S., Swanson, L.: Semi-markov dynamic programming approach to competitive bidding with state space reduction considerations. European Journal of Operational Research 32(3), 435 – 447 (1987)
71. Hasnaoui, A., Omari, A., Azzouz, Z.E.: Optimization of building energy based on mixed integer linear programming (2022)

72. Ho, T.W., Yao, J.S., Chang, Y.T., Lai, F., Lai, J.F., Chu, S.M., Liao, W.C., Chiu, H.M.: A platform for dynamic optimal nurse scheduling based on integer linear programming along with multiple criteria constraints. p. 145 – 150 (2018)
73. Hollnagel, H., Malterud, K.: From risk factors to health resources in medical practice. Medicine, Health Care and Philosophy (3), 257–264 (2000)
74. Hoseiny, F., Azizi, S., Shojafar, M., Ahmadiazar, F., Tafazolli, R.: Pga: A priority-aware genetic algorithm for task scheduling in heterogeneous fog-cloud computing. In: IEEE INFOCOM 2021 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS). pp. 1–6 (2021)
75. Huang, Z., Zhao, X., He, C., Gu, Y., Zhou, H., Zhao, B.: Fast optimal resource allocation algorithm for multicast ofdm systems (2012)
76. Imtiaz, S., Koudouridis, G.P., Gross, J.: On the feasibility of coordinates-based resource allocation through machine learning (2019)
77. Indarapu, S.R.K., Vodithala, S., Kumar, N., Kiran, S., Reddy, S.N., Dorthi, K.: Exploring human resource management intelligence practices using machine learning models. Journal of High Technology Management Research 34(2) (2023)
78. Jiang, Z., Ji, R., Chen, Y., Ji, W.: Machine learning and simulation-based framework for disaster preparedness prediction. In: 2021 Winter Simulation Conference (WSC). pp. 1–10 (2021)
79. Jo, H., Kim, Y., Lee, H., Lee, Y., Han, H., Kang, S.: On the trade-off between performance and storage efficiency of replication-based object storage. In: Chen, J., Yang, L. (eds.) Proceedings - 11th IEEE International Conference on Cloud Computing Technology and Science, CloudCom 2019, 19th IEEE International Conference on Computer and Information Technology, CIT 2019, 2019 International Workshop on Resource Brokering with Blockchain, RBchain 2019 and 2019 Asia-Pacific Services Computing Conference, APSCC 2019. pp. 301–304. International Conference on Cloud Computing Technology and Science, Institute of Electrical and Electronics Engineers (IEEE), United States (2019)
80. Johansson, R., Mårtenson, C., Suzić, R., Svenson, P.: Stochastic dynamic programming for resource allocation. Tech. rep., FOI – Swedish Defence Research Agency FOI-R–1666–SE, Command and Control Systems (September 2005)
81. Jucan, D., Tudose, L., Bojan, I.: Constrained multi-item inventory systems with quantity discounts via evolutionary algorithm. p. 1543 – 1544. Danube Adria Association for Automation and Manufacturing, DAAAM (2009)
82. Kadri, R.L., Boctor, F.F.: An efficient genetic algorithm to solve the resource-constrained project scheduling problem with transfer times: The single mode case. European Journal of Operational Research 265(2), 454–462 (2018)
83. Kai Peng, Bohai Zhao, S.X.Q.H.: Energy- and resource-aware computation offloading for complex tasks in edge environment (2020)
84. Karimi, K.: The feasibility of using opencl instead of openmp for parallel cpu programming (2015)
85. Keerthika, S., Abinaya, N., Jayadharshini, P., Ruthranayaki, J., Vasugi, M., Priyanka, S.: Enhancing soil moisture prediction through machine learning for sustainable resource management. p. 1175 – 1179. Institute of Electrical and Electronics Engineers Inc. (2023)
86. Kennedy, J.O.: Dynamic programming applications to agriculture and natural resources (1 1986)
87. Kepler, C., Blackman, W.: Use of dynamic programming techniques for determining resource allocations among r/d projets: An example. IEEE Transactions on Engineering Management EM-20(1), 2 – 5 (1973)
88. Khalaf, R.M., Hassan, W.H.: Multi-objective groundwater management using genetic algorithms in kerbala desert area, iraq. IOP Conference Series: Materials Science and Engineering 1067(1), 012013 (feb 2021)

89. Kim, Y., Ahn, S., You, C., Cho, S.: A survey on machine learning-based medium access control technology for 6g requirements. Institute of Electrical and Electronics Engineers Inc. (2021)
90. Krzywaniak, A., Czarnul, P.: Performance/energy aware optimization of parallel applications on gpus under power capping. In: Wyrzykowski, R., Deelman, E., Dongarra, J., Karczewski, K. (eds.) Parallel Processing and Applied Mathematics. pp. 123–133. Springer International Publishing, Cham (2020)
91. Krzywaniak, A., Czarnul, P., Proficz, J.: Extended investigation of performance-energy trade-offs under power capping in hpc environments. In: 2019 International Conference on High Performance Computing Simulation (HPCS). pp. 440–447 (2019)
92. Krzywaniak, A., Proficz, J., Czarnul, P.: Analyzing energy/performance trade-offs with power capping for parallel applications on modern multi and many core processors. In: Ganzha, M., Maciaszek, L.A., Paprzycki, M. (eds.) Proceedings of the 2018 Federated Conference on Computer Science and Information Systems, FedCSIS 2018, Poznań, Poland, September 9-12, 2018. Annals of Computer Science and Information Systems, vol. 15, pp. 339–346 (2018)
93. Lampoudi, S., Saunders, E., Eastman, J.: An integer linear programming solution to the telescope network scheduling problem. p. 331 – 337 (2015)
94. Lee, J.: Efficient elitist genetic algorithm for resource-constrained project scheduling. Korea Journal of Construction Engineering and Management 8(6), 235–245 (2007)
95. Li, H., Han, S., Wu, X., Wang, F.: A novel task of loading and computing resource scheduling strategy in internet of vehicles based on dynamic greedy algorithm. Tehnicki Vjesnik 30(4), 1298 – 1307 (2023)
96. Li, J.: Resource planning and scheduling of payload for satellite with plasmodium evolutionary algorithm. Journal of Convergence Information Technology 6(8), 395 – 402 (2011)
97. Li, X., Nie, L., Chen, S.: Approximate dynamic programming based data center resource dynamic scheduling for energy optimization. In: 2014 IEEE International Conference onÂ Internet of Things(iThings), and IEEEÂ Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing(CPSCom). pp. 494–501. IEEE Computer Society, Los Alamitos, CA, USA (sep 2014)
98. Liao, H., Zhou, Z., Zhao, X., Wang, Y.: Learning-based queue-aware task offloading and resource allocation for space–air–ground-integrated power iot. IEEE Internet of Things Journal 8(7), 5250–5263 (2021)
99. Liu, H.L., Wang, Q.: A hybrid evolutionary algorithm for ofdm resource allocation. p. 43 – 47 (2013)
100. Liu, H., Wang, Y.: A method for multi-project with resource constraints based on greedy strategy. p. 22 – 27 (2009)
101. Liu, J., Liu, Y., Shi, Y., Li, J.: Solving resource-constrained project scheduling problem via genetic algorithm. Journal of Computing in Civil Engineering 34(2), 04019055 (2020)
102. Liu, L., Fan, Q.: Resource allocation optimization based on mixed integer linear programming in the multi-cloudlet environment. IEEE Access 6, 24533–24542 (2018)
103. Liu, Q., Jiang, Y.: A survey of machine learning-based resource scheduling algorithms in cloud computing environment. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 11063 LNCS, 243 – 252 (2018)
104. Liu, Z., Zhou, J., Yang, X., Zhao, Z., Lv, Y.: Research on water resource modeling based on machine learning technologies. Water (Switzerland) 16(3) (2024)
105. Liu, Z., Zhou, Y., Huang, G., Luo, B.: Risk aversion based inexact stochastic dynamic programming approach for water resources management planning under uncertainty. Sustainability 11(24) (2019)
106. Liu, Z., Wang, J.: Review of interpretable machine learning for information resource management. Data Analysis and Knowledge Discovery 8(1), 16 – 29 (2024)

107. Maaroufi, F., Camus, H., Korbaa, O.: A mixed integer linear programming approach to schedule the operating room. p. 3882 – 3887 (2017)
108. Mahboob, M., Celik, T., Genc, B.: Review of machine learning-based mineral resource estimation. Journal of the Southern African Institute of Mining and Metallurgy 122(11), 655 – 664 (2022)
109. Manninen, M.: Short-term optimization method for a local energy system using modified dynamic programming. p. 3225 – 3230 (1982)
110. Mansouri, R., Pudeh, H.T., Yonesi, H.A., Haghiabi, A.H.: Dynamic programming model for hydraulics and water resources simulating and optimizing water transfer system (a case study in Iran). Journal of Water Supply: Research and Technology-Aqua 66(8), 684–700 (08 2017)
111. Manzini, R., Accorsi, R., Cennerazzo, T., Ferrari, E., Maranesi, F.: The scheduling of maintenance. a resource-constraints mixed integer linear programming model. Computers and Industrial Engineering 87, 561 – 568 (2015)
112. Martinez, J.F., Ipek, E.: Dynamic multicore resource management: A machine learning approach. IEEE Micro 29(5), 8–17 (2009)
113. Marusenkova, T., Yurchak, I.: A dynamic programming method of calculating the overlapping allan variance. Experience of Designing and Application of CAD Systems in Microelectronics (2019)
114. Matzka, S.: Using process quality prediction to increase resource efficiency in manufacturing processes. p. 110 – 111. Institute of Electrical and Electronics Engineers Inc. (2018)
115. Meixell, M.J., Norbis, M.: A review of the transportation mode choice and carrier selection literature. The International Journal of Logistics Management 19(2), 183–211 (Jan 2008)
116. Meng, G., Lai, Y., Yang, F.: An optimal bus scheduling model based on mixed-integer linear programming. p. 2200 – 2204 (2020)
117. Meurisse, A., Carpenter, J.: Past, present and future rationale for space resource utilisation. Planetary and Space Science 182, 104853 (2020)
118. Miller, G., Torres-Delgado, A.: Measuring sustainable tourism: a state of the art review of sustainable tourism indicators. Journal of Sustainable Tourism 31(7), 1483–1496 (2023)
119. Moreau, L., Queinnec, C.: Resource aware programming. ACM Trans. Program. Lang. Syst. 27(3), 441–476 (2005)
120. Müller, S.: Security trade-offs in Cloud storage systems. Doctoral thesis, Technische Universität Berlin, Berlin (2017)
121. Najeh, S., Besbes, H., Bouallegue, A.: Greedy algorithm for dynamic resource allocation in downlink of ofdma system. In: 2005 2nd International Symposium on Wireless Communication Systems. pp. 475–479 (2005)
122. Neha, Joon, R.: Renewable energy sources: A review. Journal of Physics: Conference Series 1979(1), 012023 (aug 2021)
123. Noble, N.T., Dev, Y.P., Joseph, C.T.: Machine learning based techniques for workload prediction in serverless environments. Institute of Electrical and Electronics Engineers Inc. (2023)
124. Nurcahyani, I., Lee, J.W.: Role of machine learning in resource allocation strategy over vehicular networks: A survey. Sensors 21(19) (2021)
125. Paul, J., Stechele, W., Kröhnert, M., Asfour, T.: Resource-aware programming for robotic vision. CoRR abs/1405.2908 (2014)
126. Pei, L.L., Wang, Z.X.: Application of grey dynamic programming model for optimizing regional industrial structure. p. 285 – 288 (2012)
127. Pepper, A., Tischler, N., Pryde, G.J.: A quantum autoencoder: Using machine learning to compress qutrits. Institute of Electrical and Electronics Engineers Inc. (2020)
128. Perez, G.A.C., Solomatine, D.P.: ADVANCED HYDROINFORMATICS: Machine Learning and Optimization for Water Resources. wiley (2023)
129. Pietrabissa, A., Priscoli, F.D., Giorgio, A.D., Giuseppi, A., Panfili, M., Suraci, V.: An approximate dynamic programming approach to resource management in multi-cloud scenarios. International Journal of Control 90(3), 492–503 (2017)

130. Plamondon, P., Chaib-draa, B., Benaskeur, A.R.: A real-time dynamic programming decomposition approach to resource allocation. In: 2007 Information, Decision and Control. pp. 308–313 (2007)
131. Poladian, Vahe; Sousa, J.G.D.S.M.: Dynamic configuration of resource-aware services. carnegie mellon university. journal contribution (2018)
132. Ponis, S., Aretoulaki, E., Maroutas, T.N., Plakas, G., Dimogiorgi, K.: A systematic literature review on additive manufacturing in the context of circular economy. Sustainability 13(11) (2021)
133. Popov, A.A., Lopateeva, O.N., Ovsyankin, A.K., Satsuk, M.M.: Application of greedy algorithms for the formation of the educational schedule in the higher education. Journal of Physics: Conference Series 1691, 012066 (nov 2020)
134. Quang-Hung N., Nien P.D., N.N.H.T.N.T.N.: A genetic algorithm for power-aware virtual machine allocation in private cloud (2013)
135. Quyen, N.T.P., Kuo, R., Chen, J.C., Yang, C.L.: Dynamic programming to solve resource constrained assembly line balancing problem in footwear manufacturing. p. 66 – 70 (2017)
136. Rampersaud, S., Grosu, D.: A sharing-aware greedy algorithm for virtual machine maximization. In: 2014 IEEE 13th International Symposium on Network Computing and Applications, NCA 2014, Cambridge, MA, USA, 21-23 August, 2014. pp. 113–120 (2014)
137. Rampersaud, S., Grosu, D.: An approximation algorithm for sharing-aware virtual machine revenue maximization. IEEE Trans. Serv. Comput. 14(1), 1–15 (2021)
138. Ramírez-Márquez, C., Posadas-Paredes, T., Raya-Tapia, A.Y., Ponce-Ortega, J.M.: Natural resource optimization and sustainability in society 5.0: A comprehensive review. Resources 13(2) (2024)
139. Rani, S., Charaya, S.: Review of resource allocation strategies for handoff in 5g mobile communication system. vol. 2023-June, p. 1305 – 1311. Grenze Scientific Society (2023)
140. Rani, S., Nainwal, M., Charaya, S.: Analysis of machine learning based resource allocation strategies for 5g mobile networks. vol. 2023-June, p. 1987 – 1993. Grenze Scientific Society (2023)
141. Ren, C., Wang, T., Li, X., Bai, G.: An improved adaptive dynamic programming algorithm for cloud storage resource allocation. Journal of Computational Information Systems 7(15), 5401 – 5408 (2011)
142. Ricciardi, G., Hwang, S.W.: Cost-aware dynamic resource allocation in distributed computing infrastructures. International Journal of Contents 7, 1–5 (06 2011)
143. Rivera, G., Cisneros, L., Sánchez-Solís, P., Rangel-Valdez, N., Rodas-Osollo, J.: Genetic algorithm for scheduling optimization considering heterogeneous containers: A real-world case study. Axioms 9(1) (2020)
144. Rodríguez, C.C., Romero Quete, A.A., Suvire, G.O., Rivera, S.R.: Optimization of multi-period investment planning in street lighting systems by mixed-integer linear programming. International Journal for Simulation and Multidisciplinary Design Optimization 14 (2023)
145. Rodríguez-Veiga, J., Ginzo-Villamayor, M.J., Casas-Méndez, B.: An integer linear programming model to select and temporally allocate resources for fighting forest fires. Forests 9(10) (2018)
146. Rogalska, M., Bozejko, W., Hejducki, Z., Wodecki, M.: Development of time couplings method using evolutionary algorithms. p. 638 – 643. Vilnius Gediminas Technical University (2008)
147. Rudra Kumar, M., Gunjan, V.K.: Machine learning based solutions for human resource systems management. Lecture Notes in Electrical Engineering 828, 1239 – 1249 (2022)
148. Rudra Kumar, M., Pathak, R., Gunjan, V.K.: Machine learning-based project resource allocation fitment analysis system (ml-prafs). Lecture Notes in Electrical Engineering 834, 1 – 14 (2022)
149. Saad, H.M.H., Chakrabortty, R.K., Elsayed, S., Ryan, M.J.: Quantum-inspired genetic algorithm for resource-constrained project-scheduling. IEEE Access 9, 38488–38502 (2021)

150. San Cristóbal Mateo, J.R.: An integer linear programming model including time, cost, quality, and safety. IEEE Access 7, 168307–168315 (2019)
151. Shahbaz, M., Bashir, M.F., Bashir, M.A., Shahzad, L.: A bibliometric analysis and systematic literature review of tourism-environmental degradation nexus. Environmental Science and Pollution Research 28, 58241 – 58257 (2021)
152. Sheykhi, N., Salami, A., Guerrero, J.M., Agundis-Tinajero, G.D., Faghihi, T.: A comprehensive review on telecommunication challenges of microgrids secondary control. International Journal of Electrical Power & Energy Systems 140, 108081 (2022)
153. Shi, J., Tian, H., Fan, S., Zhao, P., Zhao, K.: Hierarchical auction and dynamic programming based resource allocation (hadp-ra) algorithm for 5g ran slicing. p. 207 – 212 (2018)
154. Shin, S., Brun, Y., Balasubramanian, H., Henneman, P., Osterweil, L.: Discrete-event simulation and integer linear programming for constraint-aware resource scheduling. IEEE Transactions on Systems, Man, and Cybernetics: Systems PP, 1–16 (03 2017)
155. Shin, S.Y., Brun, Y., Balasubramanian, H., Henneman, P.L., Osterweil, L.J.: Discrete-event simulation and integer linear programming for constraint-aware resource scheduling. IEEE Transactions on Systems, Man, and Cybernetics: Systems 48(9), 1578–1593 (2018)
156. Singh, H.: Performance Evaluation of Weighted Greedy Algorithm in Resource Management. Master's thesis, University of Windsor, Windsor, Ontario, Canada (2018)
157. Singh, R.I., Lilhore, U.K.: A survey of machine learning models for water quality prediction. p. 1069 – 1074. Institute of Electrical and Electronics Engineers Inc. (2023)
158. Stickley, T., O'Caithain, A., Homer, C.: The value of qualitative methods to public health research, policy and practice. Perspectives in Public Health 142(4), 237–240 (2022)
159. Stucky, K.U., Jakob, W., Quinte, A., Süß, W.: Solving scheduling problems in grid resource management using an evolutionary algorithm. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 4276 LNCS - II, 1252 – 1262 (2006)
160. Sun, Y., Yao, P., Zhang, S., Xiao, Y.: Dynamic battlefield resource scheduling model and algorithm with interval parameters. Xitong Gongcheng Lilun yu Shijian/System Engineering Theory and Practice 37(4), 1080 – 1088 (2017)
161. Tao, Y., Sun, J., Shao, S.: Radio resource allocation based on greedy algorithm and successive interference cancellation in device-to-device (d2d) communication. vol. 2013, p. 452 – 458 (2013)
162. Tiana, X., Yuanb, S.: Genetic algorithm parameters tuning for resource-constrained project scheduling problem. In: AIP Conference Proceedings. vol. 1955 (2018)
163. Toklu, Y.C.: Application of genetic algorithms to construction scheduling with or without resource constraints. Canadian Journal of Civil Engineering 29(3), 421–429 (2002)
164. Tsakalidou, V.N., Mitsou, P., Papakostas, G.A.: Machine learning for cloud resources management—an overview. Lecture Notes on Data Engineering and Communications Technologies 141, 903 – 915 (2023)
165. Venuthurumilli, P., Mandapati, S.: An energy and deadline aware scheduling using greedy algorithm for cloud computing. Ingénierie des systèmes d information 24, 583–590 (12 2019)
166. Vink, K., Schot, P.: Multiple-objective optimization of drinking water production strategies using a genetic algorithm. Water Resources Research 38(9), 20–1–20–15 (2002)
167. Wang, H., Zhang, H.: Improved hw/sw partitioning algorithm on efficient use of hardware resource. vol. 2, p. 682 – 685 (2010)
168. Wang, L.: Proactive push research on personalized learning resources based on machine learning. p. 986 – 991. Institute of Electrical and Electronics Engineers Inc. (2022)
169. Wang, S., Meng, B.: Resource allocation and scheduling problem based on genetic algorithm and ant colony optimization. In: Zhou, Z.H., Li, H., Yang, Q. (eds.) Advances in Knowledge Discovery and Data Mining. pp. 879–886. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)

170. Wang, Y., Sun, Y., Zhang, Z.: Model training task scheduling algorithm based on greedy-genetic algorithm for big-data mining. vol. 1168 (2019)
171. Washburn, R., Schneider, M., Fox, J.: Stochastic dynamic programming based approaches to sensor resource management. vol. 1, p. 608 – 615 (2002)
172. Wen, J.H., Jiang, H., Zhang, M., Yu, X.: Application of dynamic programming in resources optimization allocation of factory production line. Key Engineering Materials 474-476, 1632 – 1637 (2011)
173. Wu, L.: Simulation study on optimal allocation model of english online teaching resources based on greedy algorithm. p. 643 – 646 (2023)
174. Wu, T., Zou, Q., Yang, Y., Zhang, X., Liu, S.: A hierarchical comb interference resource allocation algorithm based on greedy strategy and evolutionary algorithm. p. 299 – 303. Institute of Electrical and Electronics Engineers Inc. (2022)
175. Xie, F., Li, H., Xu, Z.: An approximate dynamic programming approach to project scheduling with uncertain resource availabilities. Applied Mathematical Modelling 97, 226 – 243 (2021)
176. Xing, H., Xu, L., Qu, R., Qu, Z.: A quantum inspired evolutionary algorithm for dynamic multicast routing with network coding. p. 186 – 190. Institute of Electrical and Electronics Engineers Inc. (2016)
177. Xu, B., Zhang, R., Yu, J.P.: Improved multi-objective evolutionary algorithm for multi-agent coalition formation. Journal of Software 8(12), 2991 – 2995 (2013)
178. Yakymchuk, B., Liashenko, O.: Modeling the resource planning system for grocery retail using machine learning. Communications in Computer and Information Science 1980, 288 – 299 (2023)
179. Yan, J., Zheng, J.: Open sharing of digital education training resources based on machine learning. Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST 388, 142 – 151 (2021)
180. Yang, C., Xiu, Q.: A bibliometric review of education for sustainable development, 1992–2022. Sustainability 15(14) (2023)
181. Yang, J.M., Hu, Z.H., Ding, X.Q., Luo, J.X.: An integer linear programming model for continuous berth allocation problem. vol. 4, p. 74 – 77 (2009)
182. Yin, A., Guo, Y., Tang, D.: Resource-aware fluid scheduling with time constraints for clustered many-core architectures. Journal of Physics: Conference Series 1971(1), 012090 (jul 2021)
183. Yuan, P., Zhang, T.: Mixed integer linear programming for carrier optimization in wireless localization networks. vol. 2018-January, p. 1 – 6 (2017)
184. Zhang, C., Guo, P.: An inexact cvar two-stage mixed-integer linear programming approach for agricultural water management under uncertainty considering ecological water requirement. Ecological Indicators 92, 342 – 353 (2018)
185. Zhang, X., Luo, W.: Evolutionary repair for evolutionary design of combinational logic circuits (2012)
186. Zhang, Z., Oki, E.: Joint vnf scheduling and deployment: A dynamic scenario. vol. 2022-October, p. 309 – 314 (2022)
187. Zhao, J.J., Liu, B.Y., Wei, F.X.: The application of dynamic programming in the system optimization of environmental problem. Advanced Materials Research 765-767, 3045 – 3050 (2013)
188. Zhao, Q., Yan, H., Jin, J.: Research on the most efficient use of wind energy resources in the context of carbon neutrality: Overview based on evolutionary algorithm. Mathematical Problems in Engineering 2022 (2022)
189. Zhou, Y., Liu, J., Zhang, Y., Gan, X.: A multi-objective evolutionary algorithm for multi-period dynamic emergency resource scheduling problems. Transportation Research Part E: Logistics and Transportation Review 99, 77 – 95 (2017)
190. Zhou, Z., Chang, X., Yang, Y., Li, L.: Resource-aware virtual network parallel embedding based on genetic algorithm. 2016 17th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT) pp. 81–86 (2016)

191. Zhuang, X., Jiao, H., Kang, L.: Digital management and optimization of tourism information resources based on machine learning. International Transactions on Electrical Energy Systems 2022 (2022)
192. Éles, A., Cabezas, H., Heckl, I.: Heuristic algorithm utilizing mixed-integer linear programming to schedule mobile workforce. Chemical Engineering Transactions 70, 895 – 900 (2018)
193. Özcan, E., Erol, S.: A multi-objective mixed integer linear programming model for energy resource allocation problem: The case of turkey. Gazi University Journal of Science 27, 1157 – 1168 (2014)
194. Łazuga, K., Gucma, L.: Genetic algorithm method for solving the optimal allocation of response resources problem on the example of polish zone of the baltic sea. Journal of KONBiN 38(1), 291–310 (2016)

Mariusz Matuszek is an assistant professor at Department of Computer Systems Architecture, Faculty of Electronics, Telecommunications and Informatics, Gdańsk University of Technology. His main research areas focus on intelligent distributed systems, in connection with power efficient computing. Most of his professional life is devoted to academic research, with a short interruption by an engineering position with Philips early on.

Paweł Czarnul is an associate professor, v-Dean for Cooperation Promotion and Head of Computer Architecture Department at Faculty of Electronics, Telecommunications and Informatics, Gdańsk University of Technology. His research interests include: high performance computing, distributed systems, and artificial intelligence. He is an author of over 130 publications in the area of parallel and distributed processing, including 2 books. He was a PI or participated in over 20 research, didactic or organizational projects.

Received: July 22, 2024; Accepted: October 27, 2024.

Computer Science and Information Systems 22(2):491–532 https://doi.org/10.2298/CSIS240807026R

Resource-Aware Design of an IoT Node for Use in Remote Industrial and Hazardous Areas⋆

Petar Rajković[1], Milan Vesković[2], Dejan Aleksić[3], and Dragan Janković[1]

[1] University of Niš, Faculty of Electronic Engineering
Aleksandra Medvedeva 4, 18104 Niš, Serbia
{petar.rajkovic, dragan.jankovic}@elfak.ni.ac.rs
[2] Faculty of Technical Sciences, Čačak
Svetog Save 65, 32102 Čačak, Serbia
milan.veskovic@ftn.kg.ac.rs
[3] University of Niš, Faculty of Sciences and Mathematics, Department of Physics
Višegradska 33, PO BOX 224, 18106 Niš, Serbia
alexa@pmf.ni.ac.rs

Abstract. As the Internet of Things (IoT) nodes become one of the cornerstones of Industry 4.0, they tend to be incorporated into every aspect of production automation. This paper addresses the challenge of designing low-power IoT nodes based on standardized components for deployment in remote, off-grid, industrial, and hazardous environments where energy efficiency and autonomy are critical. The proposed design integrates hardware-software co-design, replacing standard hardware setup with energy-efficient components, solar-powered batteries, and dynamic working modes to reduce energy consumption. Software elements were designed with the possibility of over-the-air updates and reconfiguration. Next, battery charging routines are optimized, and the node is integrated into a cloud-based digital twin with centralized control over the complete operation cycle. The proposed node architecture achieves an energy reduction of up to 50% and, in some configurations, reduces consumption by up to one-tenth compared to conventional designs. The additional result is a set of design recommendations when the standard components must be adapted for harsh environments.

Keywords: internet of things, resource awareness, industry 4.0, hardware-software codesign.

1. Introduction and Background

The IoT represents a world of relatively small devices connected to networks that can capture, use, and exchange data [28]. In recent years, this emerging paradigm has spread over business integration [35] and industrial automation. It created benefits for smart manufacturing [38] and Industry 4.0 [1], fueling the advances considered the new industrial revolution. Integrating IoT devices with increased computing power brought benefits not envisioned a decade ago [14]. Installing such devices to the production lines initially facilitates the data exchange with control systems. As a primary consequence, the reaction of the complete production systems becomes faster, better, and more accurate. With more

⋆ This manuscript is an extended version of the papers published in proceedings of the CERCIRAS 2023 workshop and SQAMIA 2023 conference.

extensive and detailed data sets, the production enterprises could initiate the changes in the planning process and give an additional plus to the production [36].

Our research group has designed software components for different manufacturing systems for over a decade. The research has been focused on solutions targeting the planning [5], execution [4], development [40], and deployment [41] of the software for industrial systems at various levels according to ISA-95 (ISA – International Society of Automation) standards [22]. The requirements and challenges vary from level to level, but operational efficiency is a must. The research presented in this paper focuses on ISA-95 levels 0 and 1. Levels 0 and 1 consist of sensor networks, actuators, and other devices that bring data to IoT nodes. Such nodes sometimes operate in complex and demanding environments, aiming to be self-sustainable as much as possible. In such a case, the design must consider that the device will run in harsh exploitation using minimal power and network resources.

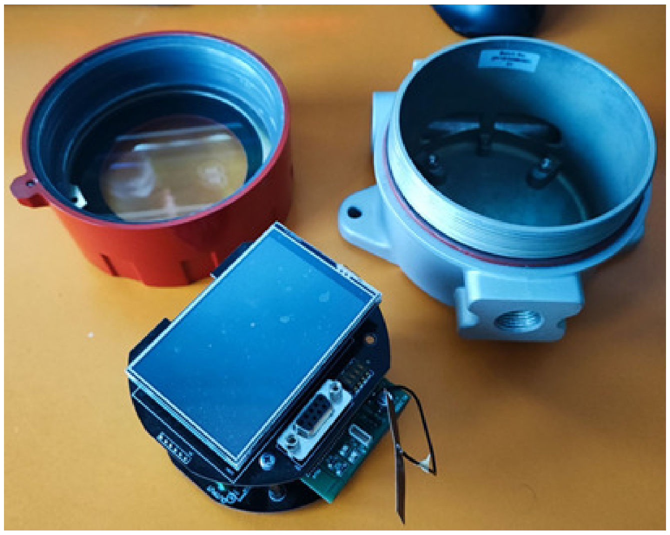


Fig. 1. Developed IoT node before sealing in the safety Ex e casing

Industrial hazardous areas, such as processing refineries, are the parts of the plants and industrial facilities where the environmental effects could permanently damage human health or threaten safety by emitting harmful gases or chemicals and where small parameter changes could cause an explosion [21]. This environment implies that any foreign object brought in (such as sensors and IoT nodes) must be designed with minimal environmental impact. In this light, any additional wiring and connection to different pieces of equipment is a source of high potential risk. Safety standards [34] imply that equipment must be packed into Ex e enclosures (Fig. 1). The complete node and all communication devices, batteries, and charge controllers should be in the verified casings (ideally in the same casing), and the node's building and operational costs should be the lowest possible.

The IoT nodes are considered to communicate with Edge computers. Since the communication between the IoT and Edge layer must be set up and maintained, selecting the wireless network will avoid additional wiring. This connection is also essential to make remote OTA (Over-the-air) configuration and management highly efficient.

An effective wireless connection is especially needed for mobile nodes in vehicles that carry dangerous or explosive materials. These vehicles need constant monitoring of the transported substance. Unlike stationary devices, mobile devices' location must be monitored in addition to all the standard values. It is essential to note that Edge computers in such a scenario are usually not in the same network or physically close, so the proper communication protocol must be defined or chosen.

To meet the requirement for such a node, we started the research that resulted in a new architecture. The architecture employs all the benefits of the IoT concepts, supported by general resource awareness. Initial results are presented as conference papers [42] and [43], and this work represents their direct extension. The focus of the work [42] was on the battery charging routines and hardware design that examines energy consumption in different working nodes. The result is the hardware setup, which should allow the IoT system to work for as long as possible.

Another founding block for this research is the modular software development approach, which was initially described in the paper [43]. Necessary changes in hardware design must be followed with new approaches in software development to make the complete system effective. Description of the IoT node's software platform, the routines for transitions to sleep mode, node update, and configuration steps are included from [43] and extended to make the complete picture of the developed IoT node.

Besides many custom-built solutions in the market and the literature, the main requirement was to stay with the widely used components, which are easily affordable worldwide and backed up by comprehensive support communities. Many existing (entirely off-grid) designs were built on high-end components that are either too expensive, not easily replaceable, or without a broad enough support network. Having in mind maintainability, together with the focus on low energy consumption, the following main design goals are formulated:

- Base the design on the standard components proven in the industrial environment to reuse standardized solutions and increase maintainability;
- Identify the top energy consumers within the standard IoT node and replace them with the appropriate external components. In this way, energy consumption should be reduced, and the maintainability level should remain the same;
- Introduce redundancy for the critical elements of the design. This will increase the system's availability and general readiness (such as transmission modules and sensors);
- Introduce new working modes for the existing IoT component – to improve system readiness and reduce energy consumption;
- Include battery charging strategies as described in [42];
- Create an easily adaptable software model that will allow node behavior change without installation or restart – to improve both maintainability and energy consumption;
- Support runtime changes of the working modes and make the system highly responsive to the update requests;
- Integrate the node into the digital twin to make the complete system more controllable.

All the requirements align with designing the IoT node with a higher readiness, better maintainability, higher stability, and lower energy consumption. This paper presents the results achieved in line with these guidelines. Section 2 represents a review of the research whose concepts were adopted and updated during the development of the IoT node. After that, hardware and software designs and energy management are elaborated in the materials and methods sections. In the section Results, measured and estimated values are compared with the expected energy levels suggested by the default designs to doc-ument used components. Ultimately, all benefits, challenges, and suggestions for further research are pointed out.

2. Related Work

This research aims to define the energy and process-efficient IoT node that should work in hazardous areas with contradictory requirements by exploring advances in hardware and software. Analyzing energy usage, the IoT node spends some power during standby, some when collecting data, some when processing them, and finally, when transmitting to the Edge level. Since energy reduction could be achieved in every step, we checked many studies to create a promising approach for the overall node design.

Study [7] advocates using power-saving modes and introducing execution cycles with multiple sleep modes. This approach constantly evolves, and [6] further introduces a complex model of sleep states where each is used in separate process steps. In [31], the advantages of decentralized IoT architecture were pointed out. We considered this concept when developing general-purpose nodes that can perform different roles by employing different software setups. This aligns with the recent findings presented in [32], where one of the main recommendations is to create IoT networks based on the lowest possible number of node types and processes.

Looking at the software side of the design, we focused on two main aspects: building a highly adaptable software model that could be easily extended and employing control mechanisms that could reconfigure real-time execution by changing the control flags. We accepted the idea behind the task allocation algorithm to reduce the time required to process the high workload in IoT [16]. The control process sets up a set of activation flags that activate only necessary parts of the processing loop in specific loops. The same principle was used when we tried to optimize the data processing routine and the size of synchronization queues in runtime.

The study [56] brought the dependency inversion principle when the driver routines for new sensors are developed. With this approach, the data collection part of the program could be developed faster and with significantly fewer changes in the complete system. Since our IoT node tends to be as general as possible, this approach enables flexibility when integrating new sensors. The mentioned work brings a complete energy-efficient framework based on several more design concepts, which could be obtained only up to some portion due to different programming paradigms used in the current software design of the suggested solution.

The contribution of the previous study is also by raising awareness of general energy consumption reduction through software design. The research [49] initially raised the attention of so-called energy bugs and hotspots resulting from the software design and scheduled task execution. Further research from the same authors [48] provides a deeper analysis of inter- and intra-task energy hotspots, with use cases and guidelines for minimizing their impact. The suggestions are integrated into the primary execution model and battery charging algorithms, similarly as suggested in [46]. They have been implemented in the presented solution by decoupling the data collection and processing from the data transmission routine.

Data transmission is critical since it uses a sizable part of the energy. The studies [13][39] give us an insight into the expected power consumption modes for the data transmission phase when different scenarios and technologies are deployed. The general suggestion is to keep transmission devices in the lowest possible energy regime as long as possible. In the ideal case, the suggestion is to keep transmission equipment in sleep mode for more than 99% of the time, regardless of the technology used.

For primary data transmission technology, LoRaWAN (Long Range Wide Area Networking) was a choice for our solution due to a higher transmission range and a longer battery lifespan compared with similar technologies [3]. LoRaWAN is usually not the first choice for data transmission. Bluetooth-enabled devices are considered a standard solution, but their limited range could not be used as communication components in the expected exploitation conditions. However, "design principles for selecting hardware components subject to varying environmental conditions and application requirements" are inherited from [23]. An excellent example of using LoRaWAN technology is presented in [26]. It describes the IoT node used in water management systems. The presented node works outdoors and has proven to use LoRa (Long-Range) technologies for its reliability and excellent power consumption rate.

The IoT nodes are intended to work as a part of a more comprehensive system, and it is necessary to define the environment that would allow fast recovery when the IoT node needs to get refreshed or reconfigured. Firstly, the set of recommendations for the software update processes in different IoT levels has been defined [5]. It was followed by establishing a digital twin structure, which was recognized as a need to support development and testing and later support when the system was in active usage [41]. During the research, dark launch expanded with feature flag deployment, which looked interesting, with the possibility of a broader application [19]. It was based on the concept that specific software features were enabled or disabled based on the value of the corresponding flags. The feature would be active only when the flag was set. The flag could be set or reset through the external interface, and the software behavior could be changed without restarting or reinstalling. We designed the ESP32 node's main loop and all other software tasks based on the feature flags approach.

The paper [25] describes a highly scalable solution that organizes IoT nodes for monitoring hazardous areas. It envisions a case where the set of static IoT nodes is active simultaneously with the set of mobile nodes and where the network can perform self-healing up to some point. The next crucial point in the research [25] is an effective alarming process. The research defines the concept of "smart alerting for potential hazard avoidance." The design rules and the algorithms for raising alarms were adapted when system parts reported problematic values, switched to backup routines, or stopped responding.

IoT nodes based on ESP32 microcontrollers whose communication part is based on MQTT (Message Queuing Telemetry Transport) protocol are proven as a choice that could support heavy computational requests. The research presented in [8] demonstrates the usage of such a combination in the system dedicated to monitoring self-generated energy during trading activities based on the Ethereum blockchain, which makes it applicable for sensor network support.

The security in such systems is not at the highest possible level, and future work will focus on this. Currently, the developed system relies on the standard security features integrated into used components and protocols. According to [30], this is assumed to be a potential security concern. Compared to other computing devices, IoT nodes have lower processing power, so specialized countermeasures against network attacks should be designed [30]. Furthermore, the research presented in [44] explains all negative aspects of the MQTT-SN (Message Queuing Telemetry Transport for Sensor Networks) protocol in detail. When it comes to energy management, the second part is charging strategies. In [9], the authors discussed traditional charging control methods, such as constant current, voltage, pulse charging, and software-enabled battery management systems. We used some principles of fuzzy logic charging as the extension of standard threshold-based charging, such as an adaptive standard low threshold. The approach presented in [9] that we found interesting is the predictive control model of energy storage systems. The study presented in [12] explains 26 different battery charging strategies. This was important to us since it explicitly focused on the charging characteristics of Li-ion batteries. It comprehensively explains controlled features, cut-off conditions, and observed parameters. The suggested multi-step-ahead predictions based on accumulated parameter values would help determine the right time to start charging. This approach was a base for our alarm-based and controlled charging scenario.

With the anticipated growth of battery management systems by more than 50% annually until 2030 [59], this research area is considered highly important and with the expected high-level improvements. This research also indicates the importance of machine learning and building an adaptive battery management system that should consider multiple parameters for their operations.

Table 1. The main features of similar solutions from literature

Feature	WaterGrid Sense [26]	E-Nose [27]	Fire detection [33]	Presented solution
Transmission protocols	LoRaWAN	LoRaWAN	LoRaWAN, GPRS, Wi-Fi, Bluetooth	LoRaWAN, GPRS, Wi-Fi, Bluetooth
LoRaWAN device class	A	Probably B, (model-based)	Probably B, (model-based)	C
Sensors	Fixed package of two sensors	N-IGSS sensor node	Various Maximum 4	Various Maximum 4
Processing unit	Microchip model non-specified	ESP32	ESP32	ESP32
Battery	3.7V 1000mAh	Non-specified	Non-specified	3.6V 3500mAh
Charging	Always when sunlight detected	Not implemented	Not implemented	Adaptive charging
Solar panel installation	External	Possible	Possible	Integrated or External

Looking at the literature, many IoT-based solutions based on a single node can be found. The most similar that we could identify are Water-Grid Sense [26], E-Nose application to detect pollution hazards [27], and forest fire detection system [33] (Table 1). All these solutions are based on LoRaWAN as the primary communication channel. The fire detection system and our solution include a GPRS module as the backup channel. Forest fire detection solutions anticipate higher energy consumption due to the higher usage rate of GPRS; thus, they work at a much higher voltage level than others. E-nose and fire detection applications did not focus on effective battery management but on higher-volume data usage. Regarding dimension, Water-Grid Sense is the smallest device, but it uses a fixed package of two sensors optimized for low consumption. It encloses a smaller battery and, as with our system, comes with a charging module. The difference in favor of our solution is that we use an adaptive charging algorithm that ensures longer battery life. At the same time, Water-Grid Sense charges the battery whenever sunlight is detected. The option of the external solar panel is available in all solutions. Water-Grid sense theoretically could use an internal solar panel as our solution, but currently, this is impossible since their casing is the smallest possible.

To create an energy-efficient IoT node dedicated to the specific setup, we had to support a complex co-design, including hardware elements, execution mode adaptation, new software design and update principles, and the definition of an adaptive battery charging approach. Referenced work exposed brilliant ideas but primarily focused on a single area of interest. At the same time, we aimed to combine all available techniques to make the IoT node as energy-efficient as possible.

3. Hardware Design

As the introduction summarized, the main direction of the design process was to create an IoT node based on standardized and worldwide available hardware components. The solution should be solar-powered, battery-based, and equipped with some wireless data emission device to integrate with higher levels. To reduce energy consumption, the IoT system should be based on a hardware platform that enables active and hibernate/sleep mode work. The node must be able to alternate working modes periodically or as the result of specific signals. In the active mode, it should periodically check sensors, read and process sensor data, and then send the retrieved values to the upper level. Further, the selected components must have enough processing power, a standardized operating system, and data storage capacity to integrate into the digital twin and enable remote diagnostics and control.

3.1. Hardware Components

The market offers several microcontrollers that could act as the core for the IoT nodes. Considering previous requirements, as the base component for the designed IoT node, the ESP32-WROOM-32 SoC module has been chosen [50]. It is widely used in industrial environments, and its modular design (Fig. 2) supports work in different operation modes defined by the states of internal components (Table 2). Its processing unit consists of two central ESP32 cores and an ultra-low-power coprocessor (ULP co-processor), which controls work in sleep mode. The ULP coprocessor is further supported with a real-time clock memory (RTC memory), primarily used for saving and keeping values during sleep mode. This memory allows active sensor data collection while two execution cores are inactive. The connectivity part of ESP32 consists of the wireless radio, Wi-Fi, and Bluetooth modules. For our design, integrated network modules were not adequate. To make ESP32 usable in the off-grid setup, these modules should be based on protocols with a much higher communication range, such as LoRaWAN and GSM (Global System for Mobile Communications). Integrated Wi-Fi and Bluetooth could be used in a production plant environment, but when it comes to the range and energy usage, they are not appropriate for remote areas. To keep the data exchange secure, ESP32 has integrated IEEE 802.11 standard security features, secure boot flash encryption, and essential power management to ensure the component's sleep mode activity. These basic features ensure enough security to be integrated with digital twins and to be updated OTA.

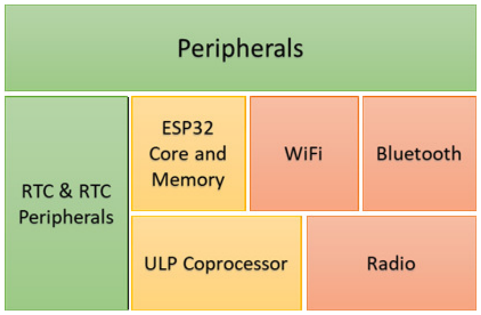


Fig. 2. ESP 32 - main building blocks

Alongside network communication components, ESP32 offers a powerful peripheral interface set that supports data collection from other hardware devices and sensors. Two interfaces are supported in this category: I2C and RS485. ESP32 natively supports I2C and comes with dedicated pins and communication routines. RS485 is a bit more critical for communication and usage in hazardous areas. It is a protocol that supports asynchronous serial communication with multiple devices and is suitable for industrial environments since it can connect to 32 devices with a cable 1200 m long. It is less prone to electrical noise.

Table 2. ESP32 - comparison of active components in standard modes

Component	Active mode	Modem sleep	Light sleep	Deep sleep	Hibernation
ESP 32 cores	+	+	paused		
RTC memory	+	+	+	+	+
ULP Coprocessor	+	+	+	+	
Radio, Wi-Fi, and Bluetooth	+				

Aside from ESP32, a few more components were necessary to complete the IoT node. The protected lithium-ion battery of type 18650, with a capacity of 3500mAh and working on 3.6V, was chosen. The battery is supplemented with a charge controller and an adequate solar panel. Supporting the battery charging process is critical for such nodes, so the chosen solar panels must be strong enough to enable successful recharge.

The complete hardware design – ESP32, battery, GSM unit, LoRaWAN module, charger, and optional solar panel- are combined as a single device and enclosed in the proper casing, certified for use in hazardous areas (Fig. 3). Since the GSM and LoRaWAN modules are used because of their range, the choice of ESP32 microcontroller was a bit challenging. In the market, many similar devices, including support for I2C and RS485, could be considered good candidates for the base component. (Table 3) shows a brief comparison of their most essential features.

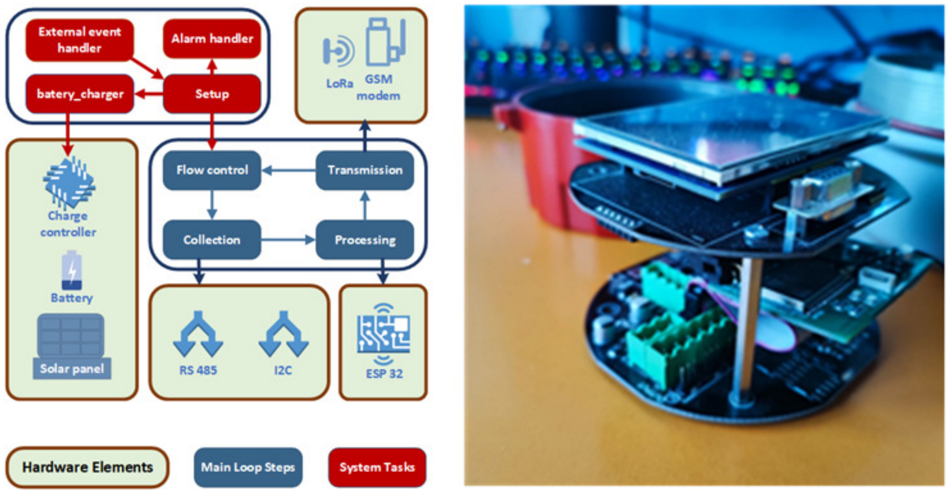


Fig. 3. IoT node for hazardous areas – left [43]: schematic display with interaction between software and hardware elements, right: the look of the assembled device

Table 3. Comparison of ESP32 and similar microcontrollers (extracted from [11])

Controller	Clock speed (MHz)	Flash memory (MB)	Maximal operating voltage	Price ratio (against ESP32)
ESP32	240	4	3.6	1
Raspberry Pi Pico	133	2	5.5	1
STM32	480	2	3.6	3
Arduino Nano	16	0.03	5	2
Teensy	600	8	5	3.5
nRF52840	64	2	3.6	2

ESP32 is one of the cheapest chipsets in the market and offers worldwide support with a strong and responsive community. There are faster components like STM32 and Teensy, but they are more expensive. ESP32 is second best in memory capacity and third in the clock speed category, but it is the cheapest and works at the lowest voltage level. In that light, it is also one of the components with the lowest energy consumption. The advantage of Raspberry Pi, STM32, Teensy, and nRF52840 is that the ARM architecture offers the base for more advanced software and hardware platforms, but with the current setup, taking into consideration all the mentioned aspects (speed, data capacity, energy consumption, and support community), ESP32 has been considered as the optimal choice.

3.2. Working Modes

The mode when all components are running is considered active, while all the other modes are considered sleep modes (Fig. 4). In active mode, the controller has maximal processing power, and all communication means are active. Consequently, it uses the most possible amount of energy and should be rarely used in configurations when energy efficiency is the primary goal.

Each sleep mode has a distinct set of active components. In modem sleep, peripherals and communication elements are disabled, while core and memory are active with the ULP processor and RTC and RTC peripherals. Modem sleep is used when the node actively collects sensor data and processes them locally without uploading them over the network. This mode had the potential for standard use but was not adopted because no external control was possible. The light sleep mode is designed to spare more energy since the core and memory are paused. It allows fast wake-up upon the signal's arrival or after the timer has elapsed. Its intended use is when the node only collects data from the sensor array.

Deep sleep and hibernate modes are intended for use when a node is in the state when waiting for the following command but with the ability to change its state as fast as possible. In deep sleep mode, RTC parts and ULP coprocessors are only active, waiting for the signals from the sensors. In hibernate mode, RTC is the only part that stays active. So, in hibernate mode, everything is shut down in the node, and the node will wake up only after a predefined time.

The working modes described are native to ESP32, and switching between them is fully supported. Since the device spares significantly more energy when in active mode, keeping the active mode as short as possible and switching between appropriate sleep modes when necessary is essential. Keeping the node in the lowest sleep mode will significantly reduce energy use.

However, for our implementation, we needed to slightly modify the mode system and introduce a new working mode – the so-called controlled active module (CAM). CAM is intended to replace active mode, modem sleep, and light sleep mode. The main idea is to switch off the complete network communication subset in ESP32 since they are not used. At the same time, the peripherals block will be kept active, allowing communication using external components and enabling the node to communicate with other pieces of software. The activity of processing cores could be controlled through the software routines, enabling fast changes in the state of active components. With this approach, the node will have processing cores active for more time compared to the default active mode for the same amount of energy.

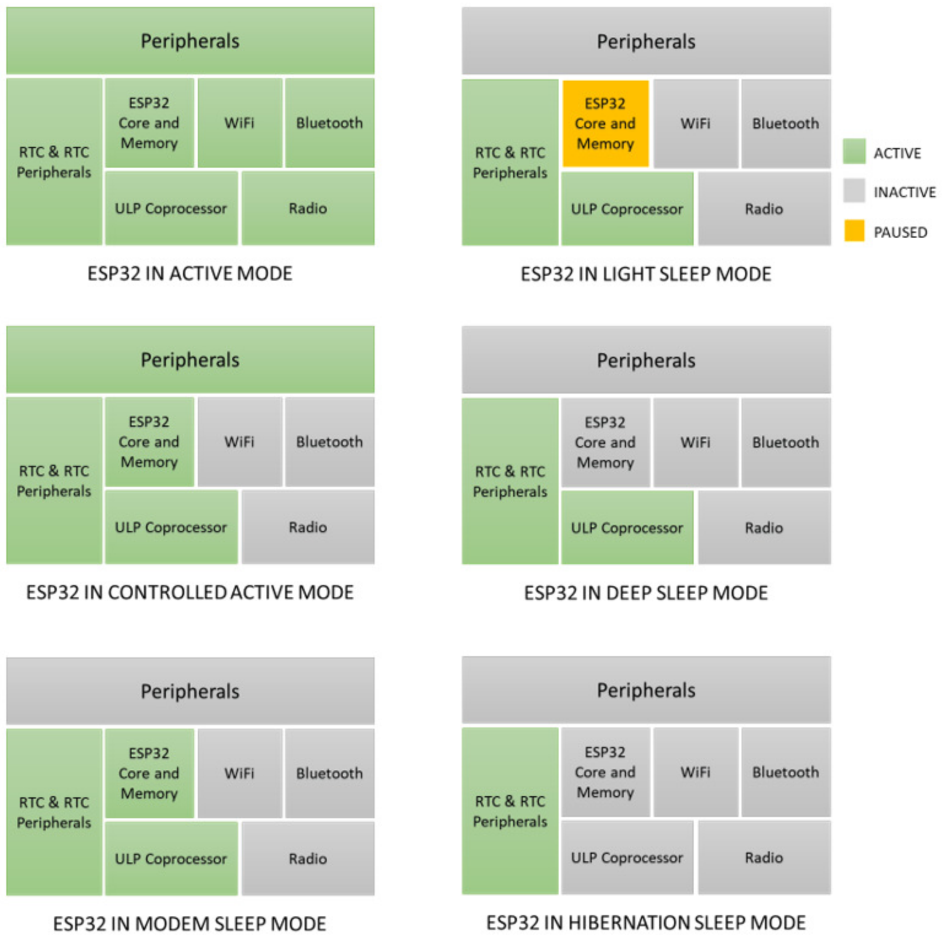


Fig. 4. Comparison of active elements in ESP32 standard working modes and Controlled Active Mode

3.3. Communication Channels

As stated before, the ESP32's communication channels had to be disabled because of limited range and high energy consumption and replaced by LoRaWAN and GSM modules. Considering all the previously described criteria, the LoRaWAN was the best fit for the design. It defines the communication on the network level and supports the protocol, which runs on the physical level and provides data exchange over long distances. Overall, the LoRaWAN technology stack positively impacts the battery lifecycle, network capacity, quality of service, safety, and security. It ensures stable bidirectional low-speed communication between mobile devices and offers the possibility to develop specialized and localized services. The data transfer speed is between 0.3 and 50kbps, which is assumed to be a compromise between the connection range and the maximum message length [54].

The main drawback is that the communication under the LoRaWAN protocol does not support data exchange between IoT nodes or other terminal devices. It supports communication between IoT nodes and LoRa gateway devices and vice versa. In LoRaWAN networks, there are three categories of node devices: A, B, and C. Only class C, or bidirectional end devices, has been considered for the presented node design. After every data package has been sent, the class C device has two short message receive time windows.

Since the IoT nodes run in off-grid areas, they must have a backup communication channel. When the LoRa channel gets interrupted or out of use, the node must be able to continue sending collected data. The backup channel was realized on a SIM-based (Subscriber Identification Module) GPRS/UMTS (General Packet Radio Service/Universal Mobile Telecommunications System) connection.

The system automatically switches to backup communication when the primary channel gets disconnected. Communication in the backup channel is much more expensive since it requires a billable connection via a mobile network operator. The added cost is related to energy consumption. The GPRS/UMTS module uses more energy for its work than the LoRa devices. For this reason, the switch to the backup communication channel is the automatic switch to the alarm state. If the main channel becomes operative again, the system automatically switches back to the LoRa connection and returns to normal operation mode.

Besides being chosen as communication channels, LoRaWAN and GPRS are slower and have lower bandwidth than Wi-Fi. This was not seen as a drawback when designing this solution since the messages carrying measured values are short. The standard packet size created by measuring probes is 26 B. Supporting two sensors and adding the necessary header makes a complete message size for one measurement of around 100 B. Also, considering that the node should not deal with real-time data but collect measurements in a matter of seconds or, more likely, minutes, low bandwidth for low energy consumption looks like an acceptable trade-off.

4. Software Design

The software component of the IoT node design is developed on top of the FreeRTOS [45] operating system. It is compatible with and supported by an ESP32 microcontroller. Its main advantage is that it fully supports multitasking, catering to the latest requirements of IoT devices.

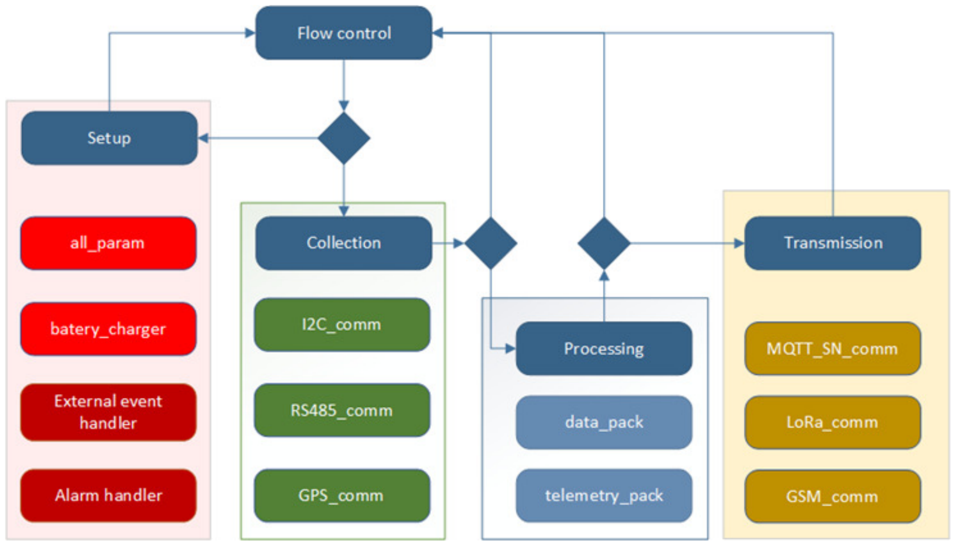


Fig. 5. Main loop and support tasks running in the realized IoT node (as in [43])

4.1. Software Processes

The software implementation of ESP32-based nodes is designed around the main task: the core revolving routine. It could call other tasks for execution, and their number is not limited. Additional tasks can either be controlled by the main task or triggered in response to specific environmental signals. The main task consists of five steps (Fig. 5), where each step calls specific tasks:

- *Flow control* is responsible for reading configurations and setting up process flags and parameters, making the main loop go only through the necessary steps.
- *Setup* facilitates the configuration of control flags and enables or disables specific aspects of the system. It is responsible for switching between execution nodes, managing the update process, and reporting data back to the digital twin
- The *collection* step manages communication with sensors and retrieves measured data.
- *Processing* is where collected data are verified and packed into synchronization objects. The created objects are then placed into synchronization queues and prepared for transmission.
- *Transmission* is when prepared synchronization objects are dequeued and sent through the network using appropriate communication.

Various tasks are implemented in every step to facilitate the IoT node's operation. These tasks fall into three main categories: setup and maintenance (indicated by red graphic elements in (Fig. 5)), data processing (light blue elements), sensor communication (green elements), and data transmission tasks (amber elements). Namely, as explained in detail in [43]:

- The *all_param* task encompasses a set of routines and data structures responsible for managing system setup parameters.
- The *battery_charger* task monitors the battery level and controls the charging procedure, ensuring the IoT node maintains sufficient power for uninterrupted operation.
- The *external event handler* is the gateway for controlling the external network. It is responsible for receiving and processing commands from the cloud or other controlling devices and forcing processes such as OTA updates, immediate battery charging, or a change of the execution mode.
- The *alarm handler* raises alarms when specific parameters reach predefined critical values. As a result of its action, the node could go to the hibernate node, or communication with a faulted external device could be terminated.
- *I2C_comm* and *RS485_comm* facilitate data exchange between the IoT node and connected sensors using one of the protocols. They ensure efficient communication and promptly support exchange routines.
- The *GPS_comm* task handles communication with the GPS (Global Positioning System) module. Accurate device positioning is crucial when the node is installed on a moving object, such as a barge transporting crude oil in rivers.
- Processing step runs *data_pack* and *telemetry_pack* processes. They are responsible for packing sensor readings (data_pack) or node's status parameters (telemetry_pack) into synchronization objects.
- The *MQTT_SN_comm* task manages the synchronization queue's capacity and occupancy. It coordinates write processes from data producers and read processes from data consumer tasks.
- *LoRa_comm* task supervises communication between the IoT node and the Edge computer using the LoRaWAN protocol.
- *GSM_comm* task oversees the backup communication channel between the IoT node and the Edge computer.

4.2. Message Protocols

Devices at the Edge level are considered much more potent than IoT nodes and can run more advanced software and communication equipment. This led to choosing the correct communication protocol focused on data delivered to the consuming Edge devices not by their network addresses but as a function of their contents and interests.

The IoT node and Edge layer communication is realized using the MQTT-SN (MQTT for Sensor Networks) protocol (Fig. 6). It is a sub-variant of MQTT modified for the wireless communication environment, characterized by low bandwidth, high link failures, and short message length [24]. Since MQTT-SN is perfected for low-cost, battery-operated devices with limited processing and storage resources, it could fully support the IoT node's hibernate mode and the LoRaWAN class C protocol.

The connection between Edge and upper levels could be fulfilled using MQTT, an open and lightweight publish/subscribe protocol explicitly designed for machine-to-machine and mobile applications [53]. The MQTT protocol is adequate since a stable wired connection connects the Edge and cloud levels. Since variants of the same protocol are used across the entire system, the whole structure has certain advantages in system response to hazardous events, overall system reliability, data security, traffic reduction in the Edge-client connection, and the background for introducing digital twins.

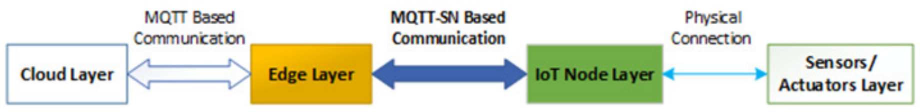


Fig. 6. Place of IoT nodes in broader ISA-95 technology stack and data exchange means be-tween layers (as introduced in [21]

4.3. Task Synchronization Mechanism

The management of configuration parameters within the FreeRTOS environment relies on established and widely recognized mechanisms. Specifically, semaphores regulate access to shared resources and effectively facilitate data exchange among tasks. To improve efficiency, the IoT node uses internal synchronization queues (set up as the internal variables in all_param tasks) between collection and processing and between processing and trans-mission steps. This way, steps that consume less energy could be performed several times before the next step, which consumes more energy, would run. With this approach, energy consumption in controlled active mode could be further reduced. As previously elucidated, the primary objective of the IoT node centers around capturing data from sensors via RS485 or I2C interfaces. Periodic data retrieval occurs concurrently through the RS485_comm and I2C_comm tasks. These tasks write data to the same message queue, guarded by semaphore. Consequently, data processing could remain dormant until the queue is filled up and only switch to an active state. Once the buffer contains enough data, the loop task proceeds with data validation and processing. The processed values are then written in the message queue for transmission to the edge level. This process is supported by I2C_comm and RS485_comm tasks. They execute concurrently and write the values they read from sensors to the same message queue. At the same time, task MQTT_SN_comm reads the items from the queue and prepares them to be sent to the cloud (Fig. 7). Using the three tasks mentioned, the semaphore approach avoids eventual read/write hazards during concurrent access to the mqtt_msg queue. Every task that should access the message queue waits until it is free and only enters the critical section. The task releases the message queue when the read or write is done, and the next task can access it.

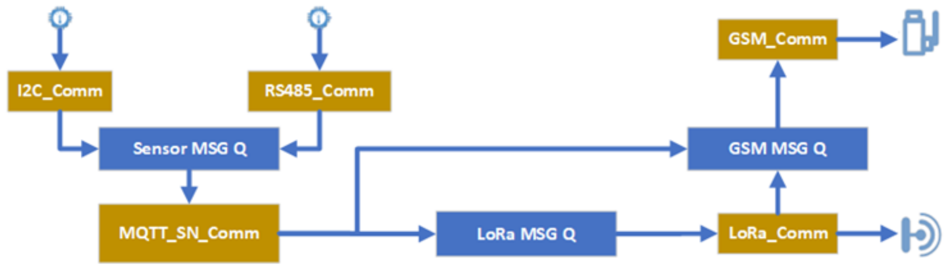


Fig. 7. Data flow from sensors to transmission elements through message queues and processing tasks

Message queues are also used for data transmission, one for LoRaWAN and another for the GSM module. The LoRa message queue does not need synchronization since each data producer has only one data producer and consumer. On the other hand, the message queue dedicated to the GSM module must be synchronized in the same way as the message queue used for data collection from the sensors. It can receive data directly from the processing step or data that failed to be sent using LoRa_comm.

4.4. Over-The-Air Update Concept

The IoT nodes are OTA-controlled from the cloud level. The complete setup, consisting of node configurations, parameters, and running software, is stored in the digital twin in the remote server. As mentioned, the IoT nodes only connect to upper levels and exchange data. In this way, the size of the IoT node network is not limited by the nodes themselves or their design but by processing power and throughput at the upper levels. Each IoT node is uniquely identified by its MAC (media access control) address, and it is initially registered in the server by the mentioned address and several more unique parameters. In this way, every communication with the node can be easily verified, and connections with potentially intrusive nodes can be easily suspended. At the same time, the server pings IoT nodes periodically and can detect these in disconnected mode. Conversely, nodes run different alarm-based notification mechanisms that inform the server about potential problems.

MQTT protocol is used with SSL/TLS (secure sockets layer and transport layer security) to enable secure communication between higher levels and IoT nodes. This feature is supported by default, but it ensures only encryption at the transmission level. To make the complete process more secure, the primary hardware level encryption is enabled in the ESP32 core, which makes all data and program structures coded, preventing different attacks, including reverse engineering, in case the code is intercepted and hijacked. To ensure the uniqueness of each IoT node, their MAC address is placed as a part of the MQTT topic. This info could be verified at the server against the registry of valid nodes.

Software updates are initiated from the cloud level, and the cloud server pushes adequate software updates to the specific IoT node. This way, the node will receive proper software and continue working after the update. This approach allows for quickly replacing the software of an IoT node and changing its role without the need to make direct changes or configuration in the node.

5. Battery Charging Routine

An ideal energy consumption scenario involves standardized functionalities that maintain consistent energy usage levels over an extended period. However, practical constraints often prevent such ideal conditions [12]. As previously discussed, different data transmission devices exhibit significant variations in energy consumption. For instance, scenarios involving updates or lost connections to sensor devices result in increased energy usage beyond the baseline. Furthermore, distinct active and sleep modes consume varying amounts of energy depending on the volume of workload nodes have to perform. Also, transitions between modes can trigger consumption peaks if specific initialization procedures are required. As outlined earlier, energy usage during node operation depends on the working mode and the frequency of necessary actions.

When evaluating data usage across the three phases of the node's cycles, data processing and data collection use a similar amount of energy. Compared to data transmission, data collection and processing use much less energy. Data transmission modules exhibit substantial differences in range, speed, and data package volume, but in any case, data transmission remains the most demanding energy task [59][27][33][17]. The battery's energy level should always be adequate to ensure proper node operation fitness. For this reason, a separate set of routines is developed and integrated into the IoT node's software model. It is intended to drive the charge controller and execute chosen charging strategies.

5.1. Automatic Charging

The charging process periodically checks the battery's energy level in the automatic charging mode. It starts if it reaches a standard low battery level (SL). The node continues its operation while the battery is charging, and when it reaches a standard high level (SH), the charging process stops. The charging controller is a separate component and does not affect the work of any other IoT node element. This approach could be problematic when the node's charging routine depends on solar power. Sunlight is available at most 50% of the time, and the periods of active sunlight are not constant. Furthermore, the effect of the other natural elements and construction properties of the device could reduce the period of sunlight exposure.

Whenever the charging controller starts or stops the charging process, it sends this information to the edge level using the telemetry call with a timestamp. These data are collected at higher levels and used to analyze node functionality and act as a base for future improved charging modes. They could also be used to identify malfunctions early. The default charging process, if applied constantly, is envisioned to ensure longer battery life. The best use case for most available battery types is if their power level varies between SL and SH thresholds, following the process presented in Fig. 8.

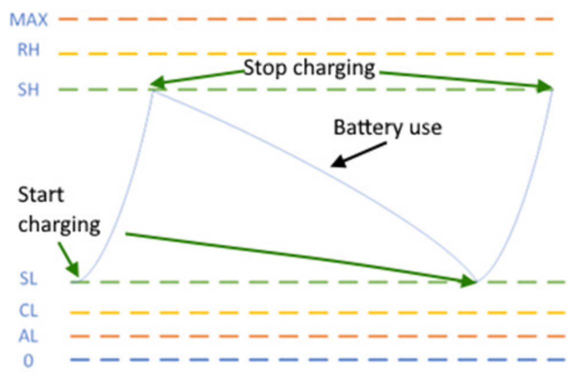


Fig. 8. Ideal consumption setup with automatic charging mode

5.2. Alarm-Based and Controlled Charging

An automatic charging scenario is not always possible. First, it could be triggered at night or when the sunlight is not bright enough. Then, the solar panel will not generate enough power to raise the battery's energy level. When charging starts, but the energy level is still going down, the alarm signal from the IoT node will trigger. The signal will be received and registered at the edge level. Since the charging controller frequently reads the battery's energy level, it could continue to trigger alarms that indicate that the energy level is still reducing despite initiating the charging process (Fig. 9, block "Report charging issue"). If the energy level continues to reduce, it will eventually reach CL (Charging Required Level). At that moment, the IoT node will send a higher priority alarm to the Edge computer and reconfigure its operation strategy by reducing the number of data transmission operations. If the battery level continues to degrade, after some time, it will reach the alarm low (AL) threshold (Fig. 10, left). This is considered the highest-level alarm, and the node will stop all its operations and switch to hibernation sleep mode. Up to that time, based on the data received in the Edge and then forwarded to the cloud level, the operation engineers could decide what to do with the affected IoT node.

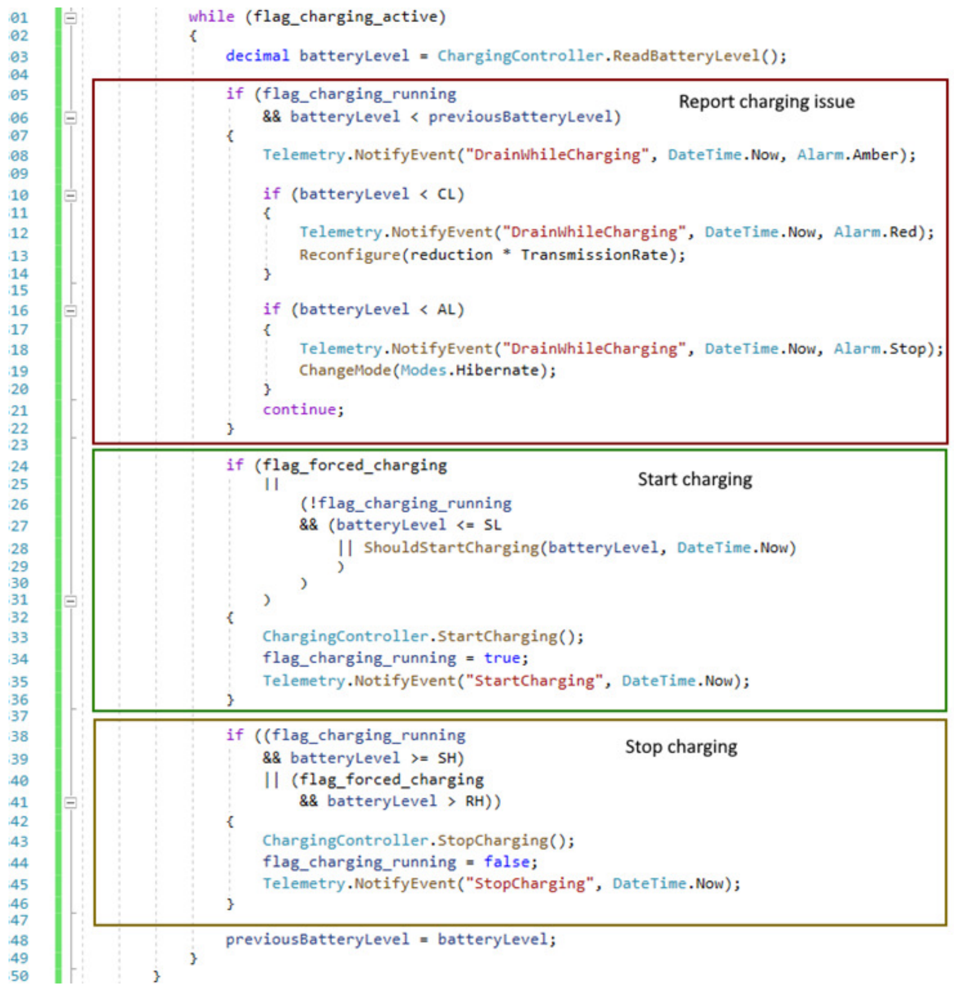
One of the simplest ways to prevent this situation is to enable the calculation of the energy use depending on the time of the day and the introduction of an additional method that will check if the charging process should start (Fig. 9, block "start charging," line 28). SL would be increased by some percentage (like 10 or 20%). In this case, the charging routine will check the remaining time until sunset and the increased SL. If the energy level falls to SL+10% and the remaining period of the day is, i.e., 10% sunlight, the charging process will start immediately. This simple and effective approach allows for additional charging periods with the lowest possible effect on battery life. The problem with such an approach is that the node must have daily information about sunrise and sunset and run more complex checks.

The charging controller's next operation mode is the controlled mode. This mode is initiated from the edge level and intended to instantly trigger the charging process. Regardless of the current battery level, the charging process will start immediately when the control signal is received, and the *flag_forced_charging* is set.

The mentioned control signal is followed by the requested high level (RH in the further text); the battery will be charged until the requested level is reached, regardless of the value set for SH (Fig. 10), suitable; (Fig. 9), block "stop charging," line 41). This process does not change the SH level but is omitted during a single charging run. When the battery level reaches RH, the charging process stops, and the node returns to the alarm-based mode. The battery could lose power in the controlled charging mode, as in the automated charging mode. In this case, the same alarm procedure will run. Eventually, the charging controller could be disabled by setting *flag_charging_active* to false. This happens regularly when the IoT node is connected to the power grid, but this situation is outside the scope of our paper.

5.3. Short Term Improvements

As explained, making the charging process more adaptive and efficient is essential. Considering that the transition to controlled charging mode with the predefined RH could be triggered from the higher levels at any time, bringing a dose of safety, the process will be



```
while (flag_charging_active)
{
    decimal batteryLevel = ChargingController.ReadBatteryLevel();

    if (flag_charging_running
        && batteryLevel < previousBatteryLevel)
    {
        Telemetry.NotifyEvent("DrainWhileCharging", DateTime.Now, Alarm.Amber);

        if (batteryLevel < CL)
        {
            Telemetry.NotifyEvent("DrainWhileCharging", DateTime.Now, Alarm.Red);
            Reconfigure(reduction * TransmissionRate);
        }

        if (batteryLevel < AL)
        {
            Telemetry.NotifyEvent("DrainWhileCharging", DateTime.Now, Alarm.Stop);
            ChangeMode(Modes.Hibernate);
        }
        continue;
    }

    if (flag_forced_charging
        ||
            (!flag_charging_running
            && (batteryLevel <= SL
                || ShouldStartCharging(batteryLevel, DateTime.Now)
                )
            )
        )
    {
        ChargingController.StartCharging();
        flag_charging_running = true;
        Telemetry.NotifyEvent("StartCharging", DateTime.Now);
    }

    if ((flag_charging_running
        && batteryLevel >= SH)
        || (flag_forced_charging
            && batteryLevel > RH))
    {
        ChargingController.StopCharging();
        flag_charging_running = false;
        Telemetry.NotifyEvent("StopCharging", DateTime.Now);
    }

    previousBatteryLevel = batteryLevel;
}
}
```

Fig. 9. Charging controller routine incorporating alarm-based and controlled charging (pseudocode)

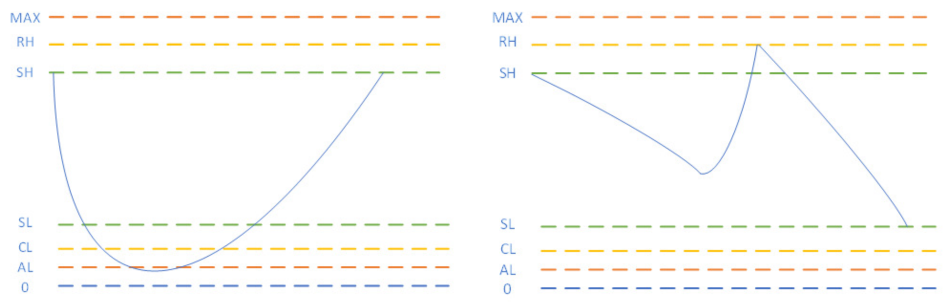


Fig. 10. Battery recharge after the intensive drain (left) and the battery charging in controlled mode (right)

automated to ensure less frequent (ideally never-happening) situations when the IoT node goes to the alarm state. The charging controller regularly reads the battery status and uploads (and stores locally) these data for further analysis. The average energy consumption per hour (ACH) is calculated based on this. Since the node reads data from the sensors during standard periods, the actual energy consumption could be an additional input for deciding when to start charging.

The next improvement will be for the method running in the node that decides when to start charging (Fig. 9), block "start charging," line 28). The update method will calculate the sum of SL and the value resulting from multiplying ACH by the number of hours until sunrise. If this sum is higher than the battery's current energy, the charging process could start immediately, significantly reducing the risk of the transition to the alarm state. Further improvements would include the weather report and checking if the potential period with less sunlight is ahead. This way, the charging process could run up to a higher threshold than SH, bringing the battery a higher operational period.

It is important to note that the charging frequency depends on the battery capacity and the effectiveness of the solar panel and the charging component. With the standardized working mode, with two RS485 sensors attached and a LoRa module used for data transmission, our node will need one charge weekly or bi-weekly. This period is long, and the weather could change several times. Also, if there is a need to use more expensive energy, GPRS communication channel energy will be drained much faster. Thus, the possibility to react fast and run charging is a necessity.

6. Results

The proposed solution is based on the ESP32 series of devices with added communication and power supply components (Fig. 11). The node is designed to be robust from the physical perspective, with easily reconfigurable hardware execution modes, and flexible from the software design point of view. Operationally, it should run using the lowest possible amount of energy while acquiring data from different interfaces. Since the system has not only the ESP32 but also other components, the measurement must be done in correlation with the entire system, not only the processor itself. The overall power consumption combines the consumption of sensors (the setup with two RS485 inductive distance sensors with a maximal 10Hz measuring rate), ESP32, and internal and external communication modules. The usual test was with 100 execution setups daily.

The measurement has been performed in the laboratory and simulated field conditions. The measured objective was the water level in the water tanks. We tested energy consumption in the laboratory with regulated temperature settings. In the simulated field conditions, we mainly tested battery charging routines. Simulated field conditions were performed at the rooftop of the Faculty of Sciences, Niš, Serbia, where solar exposure is somewhat average for Southern Europe – between 1.5 kWh/m2 in January and 6.5 kWh/m2 in July [37]. Since solar panels are usually certified for 1kWh/m2, the node is usually charged with the nominal current. The node ran constant readings from the sensors while the data processing and transmission frequency were controlled from the Edge computer. The node is automatically reconfigured when the battery level reaches critical values. Digital multimeters GDM-8255A [2] were used as measuring equipment in the laboratory, and UNI-T UT71C [57] for the fieldwork.

6.1. ESP32 Default Energy Levels

The default energy consumption data can be found in the related product datasheet [50]. The consumption analysis started with the measurement for the node based entirely on ESP32, where its internal communication modules are used. The software part is equal in this and the setup with the external communication modules, so the execution mode is assumed to be constant in the system. Internal modules are used only for the testbench since they are unsuitable for remote areas.

Table 4. Expected values for energy consumption in ESP32-based nodes (extracted from [18])

Power mode	Description	Typical power consumption
Power off	CHIP PU is set to a low level; the chip is powered off	0.1 μA
Hibernation	RTC timer only	5 μA
Deep sleep	From only RTC timer + RTC memory to ULP co-processor is powered on	10 – 150 μA
Light sleep	ESP32 core is paused	0.8 mA
Modem sleep	ESP32 core is powered	Slow speed:2-4 mA Normal speed: 20-25 mA Max speed: 30-50 mA
Active (RF working)	Receive - Transmit BT/BLE	95-130 mA
	Transmit 802.11g	180 mA
	Transmit 802.11b, OFDM 54 Mbps	190 mA
	Transmit 802.11.b, DSSS 1 Mbps	240 mA

The values shown in (Table 4) represent standard energy consumption levels measured in laboratory conditions and vary by some percentage compared to the values from the producer data sheet. Furthermore, some additional differences could be introduced due to the influence of connected sensors. In the examined case, the node was connected to different RS485-based sensors (Fig. 11).

6.2. Measured Values

As mentioned in the introduction, the opposing requirements for the designed nodes are that they should be as ready as possible and use the lowest possible amount of energy. In an important event, the node must immediately wake up, raise an alarm, and take the necessary action. Deactivating the data transmission part is how to keep the ESP32 active but use less power. This will not affect data processing and sensor connectivity, but the consumption will be lower in CAM mode, as defined in 3.2. With the new working mode, the node will be active in remote areas with lower power consumption than standard active mode and modem sleep. The complete execution setup includes switching between sleep modes and the CAM mode.

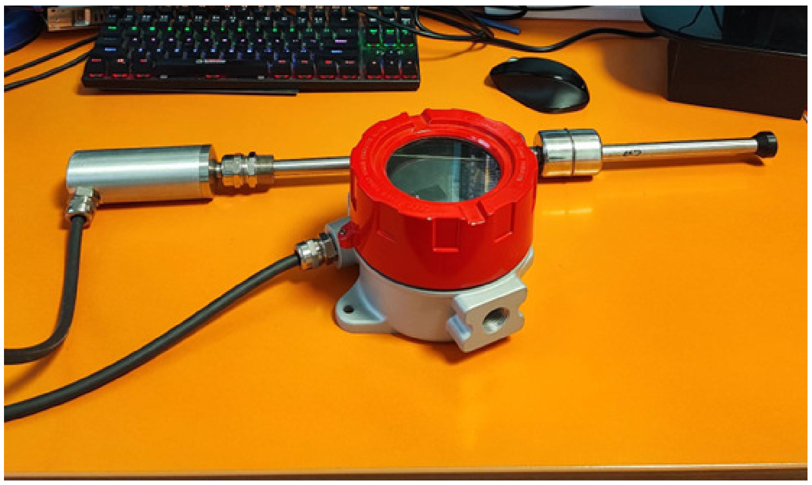


Fig. 11. Finalized IoT node with one RS485-based sensor attached

As seen from Table 5, if the standard active mode were used, the lowest possible consumption would be at least 100 mA. The power consumption in CAM mode was up to 36 mA, while the modem sleep with active processing cores worked between 45 and 50 mA. This means that CAM mode could successfully replace parts of the processing routine where both active and modem sleep modes are running. The measured values for modes with active processing outdoors were close to lab measurement, with a difference of not more than 10%.

The subsequent measurement is to connect sensors and measure the energy spent for data collection and processing at once. The sensors are connected to ESP32 through the RS485 interface. In this case, the total measured power consumption in CAM mode is 69 to 72 mA. The active components are ESP32 and two RS485 sensor arrays, whose consumption level is a maximum of 20 mA per sensor. In this case, the computed consumption was 36 + 2x20 = 76 mA. Still, the measured values remained around 70 mA in the laboratory and just above this level in simulated field conditions (72 average, 78 mA max). Compared to standard ESP32 active mode, the difference is significant, where consumption is usually 150-160mA but could hit 200 mA if unoptimized software loops are used.

Table 5. Comparison of measured values for the IoT consumption (in mA, Setup A – improved design with CAM and external communication modules, Setup B – design relying only on ESP32 internal modes and modules)

Process	Operation setup	A lab	A field	B lab	B field
Light sleep and sensors	Light sleep ESP32 core is paused	7.5	8.4	7.8	8.5
Data processing (active mode)	Setup A: CAM Setup B: Active mode	32	36	> 100	> 100
Data processing (modem sleep)	Setup A: CAM Setup B: Modem sleep	32	36	50	50
Collecting and Processing	CAM/Active mode + 2 RS485 Each RS485 < 20 mA	69	72	149	160
Transmission (worst case)	Setup A: GSM Setup B: Wi-Fi DSSS	480	412	270	240
Full cycle (standard case)	Setup A: CAM + Sensors+ LoRaWAN Setup B: Active Mode + Sensors + Wi-Fi	98	104	200	200
Full cycle (worst case)	Setup A: CAM + Sensors+ GSM Setup B: Active + Sensors + Wi-Fi DSSS	560	524	430	430

The collection-only scenario was checked when the ESP32 was put into light sleep mode. The node in light sleep mode with attached sensors uses around 8 mA regardless of the scenario. The measurement in field conditions shows an average energy need of less than 10% more. In the period when the node needs to perform data collection periodically, light sleep mode is the logical choice. The ESP32 core and memory will be paused, but with RTC components active, the node can react to requests. The consumption in light sleep mode is as low as 7.5 mA with a peak value of 8.5. The consumption of the ESP32 itself is about one mA (0.8 mA as per documentation), but, simultaneously, the battery should also power sensors on stand-by, thus the difference.

The following important measurement is the consumption level when all cycle elements run – data collection, processing, and transmission. In a setup with only ESP32 components as the transmission device, the Wi-Fi in SoftAP (software-enabled access point) or STA (station) mode is enabled. In this case, the total consumption reaches 200 mA (compared with 190 mA from documentation). The usage is at the expected level, yet another argument for using the CAM is against using the full active mode as much as possible. So, from the calculation, it could be concluded that the communication part of the ESP32, in the measured case, uses energy equivalent to 110 mA.

LoRaWAN is the communication carrier for complete cycle measurement with CAM mode. Specifically, as the communication part of the LoRaWAN module, SX1268 [47] was installed. It uses 22 mA for data transmission and five mA for data reception. As mentioned, the LoRa works in class C since the node must operate in active and sleep modes. The measured value for the LoRa communication, when data are taken from the message queue and emitted, is at the level of 28 mA for transmission and 6.4 mA for reception. The overall energy used when the complete cycle is active with the LoRa part is around 100 mA, significantly under 200 mA, measured if Wi-Fi was running (Fig. 12).

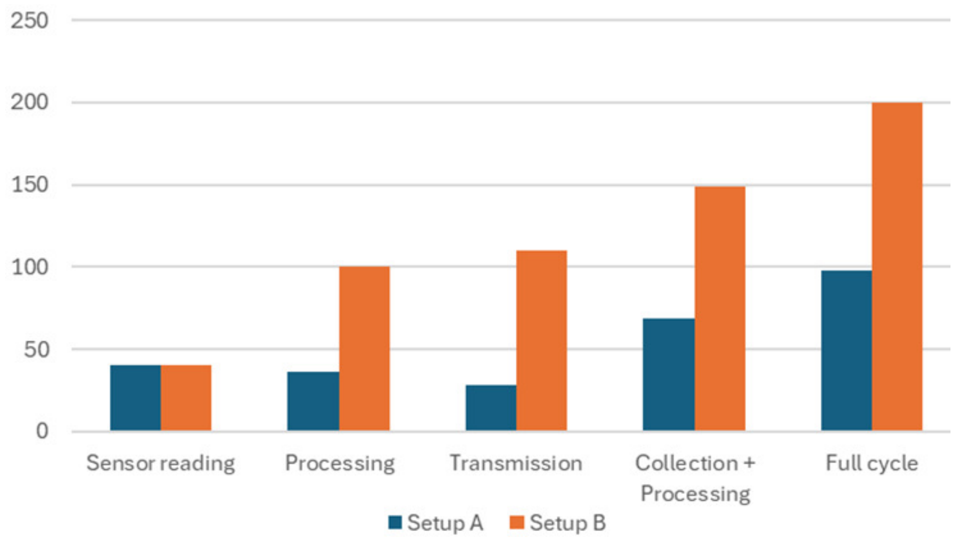


Fig. 12. Comparison of energy consumption for proposed (Setup A) and standard (Setup B) configurations

In the case of regular use, the LoRa is more efficient than internal communication modules. In urgent cases, the system needs communication to contact the device outside the internal network. LoRaWAN or integrated Wi-Fi and Bluetooth will not be helpful when the communication is broken down. The GSM module is introduced to manage such an event. The consumption of the GSM module is significantly higher than anything else, and the maximal measured level in field condition was 412 mA (345 mA as in specification) when active and 21 mA when idle (19 mA as in specification). Measured values in the lab were higher (around 480 mA) because connection establishing takes longer. The used GSM module is SIM800H [51] with GPRS data mode (1Rx, 4Tx) on EGSM900. Measured values are higher than specified but in the acceptable ratio. Setting up the connection could be the critical point in both LoRaWAN and GPRS data modules. It could take some time to execute, and the power consumption could be high during that period. The average of the GPRS module was 580 mA, while the theoretical peak could reach even 2000 mA. This fact is one of the reasons why introducing message queues and reducing the number of data transmission calls (when possible) is also essential.

When checking the complete cycle consumption with GSM, the average values are much higher than in any other setup. It was up to 560 mA in the lab, while outside reaches almost 530. Compared to GPS, the energy used in configuration with Wi-Fi running in DSSS mode was not more than 460. This is the only category where process-level updates do not bring benefits since the transmission part uses way higher amount of energy. This case clearly shows the importance of message queues and reducing transmission calls. The transmission mode could be adjusted to shrink the drawback of GPRS data module usage. Since the GPRS could manage a higher data volume, the system could decrease the number of transmissions and thus reduce overall energy consumption.

6.3. Consumption Analysis for Different Execution Modes

Measuring the energy consumption for the different elements of the IoT node offers a realistic overview of the energy consumption reduction rate. These values could also estimate energy consumption for various system configurations. By employing buffers, the number of data processing and transmitting operations would be reduced, positively impacting the consumed energy level. Table 6 and Fig. 13 show proposed energy-saving configurations and maximal measured values for every step in the process that will be used for the estimate. In this case, the measurements have been done only in the laboratory.

Table 6. Maximal measured values (in mA) for every step in the node operation

System configuration	Sensor reading	Sleep1	Processing	Sleep2	Transmission	Sleep
A+LoRaWAN	40	-	36	-	28	8
A+WiFi	40	-	36	-	110	8
A+GPRS	40	-	36	-	412	8
B+LoRaWAN	40	-	36	8	28	8
B+WiFi	40	-	36	8	110	8
B+GPRS	40	-	36	8	412	8
C+LoRaWAN	40	8	36	8	28	8
C+WiFi	40	8	36	8	110	8
C+GPRS	40	8	36	8	412	8

The execution modes are named A, B, and C. The difference is in the usage of message buffers. In execution mode A, there are no buffers. Each data collection is followed by data processing and transmission. Operation mode B introduced a buffer before data transmission. This means the node will read the data, process them, and put them into the queue. Data will be sent to the Edge level when the queue is full. Execution mode C is the update of mode B and brings an additional buffer between data collection and processing.

The maximal measured value for the sensor reading segment was close to 40 mA, which was used as the estimation value. For the processing part, the baseline value of 36 mA was considered, while all sleep modes were calculated as having the top consumption level of 8 mA. Transmission rates were acquired as 28 mA for the LoRaWAN module, 110 for Wi-Fi, and 412 for the GPRS external module.

The primary operation mode (Fig. 13, A) is the sequence read-process-transmit followed by the sleep period. Depending on the current process or state of the overall system, the node could go either in the CAM or light sleep mode. This way, the node does not need to store any data locally and can go to sleep mode at the lowest cost possible.

Since the part of the process that consumes a considerable amount of energy is the transmission part, introducing a buffer before sending data to the Edge level brings the best gain. The node would wake up periodically, read sensor data, process them, and store them in the internal buffer (Fig. 13, B). This will reduce the number of data transmissions every cycle. This is especially important when using the GPRS module since its connection setting-up part could quickly drain the battery. Note the difference in setup A with GPRS when the measured value of 61600 mA was much greater than the estimated 49600. It is partly due to indoor conditions, but the consumption is significant. More than five times compared with LoRaWAN and about 2.5 times with Wi-Fi.

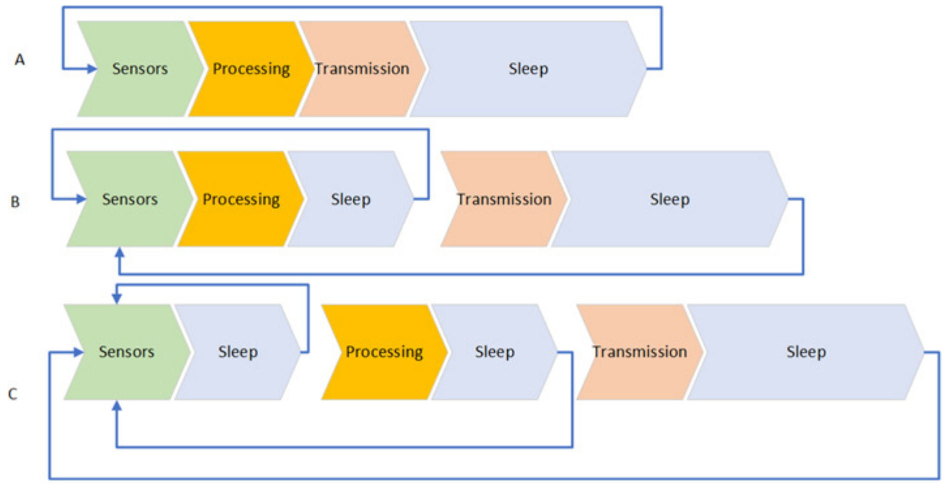


Fig. 13. Different configuration variants supported by IoT node, derived from general state-based energy consumption model

With the buffer introduced between the data collection and processing parts (Fig. 13, C), sensors will read data periodically, pump them to the message queue, and the system will transit to sleep mode. After several iterations, the processing part will get activated. It will take the data from the queue, process it, and then store it in the queue before transmission. Data transmission will run when enough data gets stored in the second queue.

The analysis was based on 100 complete work cycles to provide a more comprehensive overview of the proposed solution's expected effect. The energy usage was lowest when the configuration variant C was applied, and the LoRaWAN was used as the communication module. The worst case from the energy consumption point of view was when strategy A was applied, and the GPRS was used for data transmission.

A comparison between these three variants is shown in Table 7. The estimate was calculated on the base of 100 sensor reading cycles. Comparing one variant, it is evident that the lowest consumption is in configuration with the LoRaWAN as a transmitting device. The difference is more significant in variant A than in B and C. The number of total transmissions is in direct proportion to the energy use, so the best effect is with the default operation mode. In variant A, the system with the LoRaWAN uses slightly above one-half of the energy used by the system with the ESP32 native Wi-Fi (50.64%). The energy usage is the highest with the configurations with the GPRS transmitter. Variant A uses more than five times more energy than the configuration with the LoRaWAN and more than 2.5 times more than the native Wi-Fi transmitter.

Variants B and C have the most significant effect when the GPRS is used. Since the amount of time required for data acquisition is always the same, the number of data transmissions in variant B is reduced. In contrast, in variant C, further reductions are achieved by joining the processing part for 10 data acquisitions. In that way, in variant B, the data are transmitted only ten times for 100 reading cycles, and in variant C, only once. Variant C brings the most minor differences between configurations with different communication modules. It is on the level of 10% (107.09% vs 97.87%). This difference is almost 50% (138.42% vs 88.47%) for variant B. In variant C, the configuration with the GPRS uses less than one-tenth (9.38%) of energy compared to variant A. For the Wi-Fi as the transmitting module, the energy usage is reduced to a quarter (23.16%); for the LoRaWAN-based configuration, it is close to half (44.76%).

Table 7. Effects of proposed node configuration variants equivalent to 100 cycles measured against native communication setup (WiFi as a part of ESP32 Radio: Comp1) and against default configuration variant (A: Comp2)

System configuration	Estimated (mA)	Measured (mA)	Transmission count	Comp1 (WiFi)	Comp2 (variant A)
A+LoRaWAN	11200	11800	100	50.64%	100%
A+WiFi	19400	23300	100	100%	100%
A+GPRS	49600	61600	100	264.38%	100%
B+LoRaWAN	8760	8820	10	88.47%	74.75%
B+WiFi	9580	9970	10	100%	42.79%
B+GPRS	12600	13800	10	138.42%	22.40%
C+LoRaWAN	5276	5282	1	97.87%	44.76%
C+WiFi	5358	5397	1	100%	23.16%
C+GPRS	5660	5780	1	107.09%	9.38%

This proves that buffer use is effective whenever possible, which means that the delay of transmitted data is not problematic for the entire system's efficiency in every case. By adjusting the count of cycles in the digital twin and pushing the update to the end node, the energy consumption could be adjusted in the node without physical access.

7. Discussion and Future Work

The primary purpose of the proposed system is to run in a remote and hazardous area as efficiently as possible. The system must operate on batteries and use every opportunity to reduce energy usage. To achieve this goal, the following set of improvements was realized over the standardized ESP32-based IoT node:

– The new active working mode will be introduced by disabling modules that consume high energy values.
– Define the transition to the adequate sleep mode, depending on the node's usage cycle stage.
– Add external communication components that are more suitable for the expected use and have lower energy consumption.
– Enable redundancy whenever possible to make the system more dependable.
– Create an adaptive software model that will allow easy reconfiguration of the system's working mode without needing restart or hardware replacement.
– Introduce data buffers between system segments and make the operation of the more significant energy consumers less frequent.

Having in mind the requested purpose, the designed IoT node must be not only energy efficient but also highly dependable. It should be able to adequately supervise various errors, failures, and technical problems. Hardware and software design modifications were implemented during the proposed node's work. Hardware-level interventions are mostly related to the installation of redundant parts – both sensors and communication lines. In that sense, the IoT node has two I2C and two RS485 communication channels, while the transmitting device based on the LoRaWAN is backed up with the GPRS module.

Regarding future improvement, the widest open point is data security. ESP32 runs with integrated IEEE 802.11 security for IoT nodes, but it has been proven that this level is not enough in every case. So, improvements in this area would be one of the future research directions. For the moment, an additional security measure is that access to IoT nodes is possible only through the Edge level or, in exceptional cases, through a device that has an authentication token provided.

The effect of the implemented updates is presented in Table 5. The node's power consumption is closer to modem sleep than active mode. This is expected since the communication part uses a massive portion of energy. With sensors enabled, measured consumption is around 70 mA, which is between one-half and one-third of the consumption when the ESP32 is active. When the complete system is operational, the consumption of the designed IoT node is about one-half compared to the node running on the ESP32 in fully active mode (98 mA vs. 200 mA).

Improvements to the rest of the system are made at the software level. The crucial point was the implementation of setup routines that could directly influence the behavior of the main loop and change the execution variant of the node only by setting the feature flags. The control over these processes was moved to the cloud to create a digital twin. From this point, the updates could be directly passed down to the IoT nodes through the Edge computer. In that way, the control is centralized, and the status of each node will be successfully kept on the cloud.

Thanks to this feature, the node can easily switch operation modes and return to a more energy-efficient configuration. In variant A (Fig. 13), the node runs the collection-processing-transmitting sequence followed by the sleep period. In this mode, there is no need to store the collected data locally since they are once uploaded to a higher level. This mode uses the highest energy value but ensures the exact data reporting process.

Configuration variant B is intended to reduce the number of data transmissions, but it cannot be used in every case. It could be used only when the acceptable delay between data retrieval and transmission is long enough. The highest gain of this approach is when the GPRS data transmission method must be used since it consumes a significant amount of energy while setting up a connection to the network. Configuration variant C is the best solution from the point of view of energy consumption, but it brings additional limitations. First, the time until data are uploaded to the Edge level is even higher. Second, since the data processing part does not follow every data collection, there is some risk that potentially wrong values could be discovered later than in cases B and C. In the end, sometimes, IoT nodes must be on constant alert and run actively as much as possible. Since the consumption in fully active nodes is far from acceptable, one solution for the ESP32-based systems is the introduction of CAM when only radio, Wi-Fi, and Bluetooth are disabled. In that way, the system could stay in an active state longer and use less energy. The working mode would be the most like configuration variant B in this case.

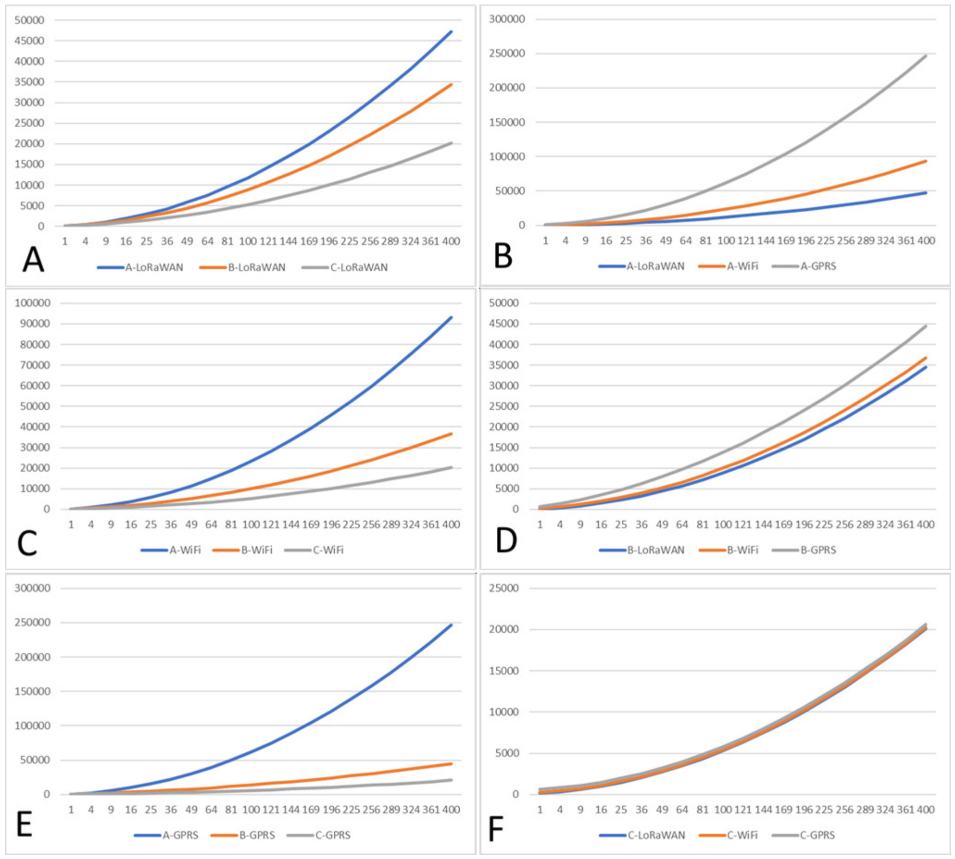


Fig. 14. Comparison of energy consumption across a different combination of variants and communication devices (The X-axis represents the number of data collection events from sensors, and the Y-axis is energy consumption in mA)

As can be seen, each of the three working modes has advantages and disadvantages, and the operation mode would probably need to be adjusted during the node's life cycle. The possibility of changing the node behavior through the software interface would help in this case. The use of the mentioned digital twin is crucially important here. The end user could adjust node behavior in the digital twin, run the simulations on data transfer and energy consumption, and then push the change to the actual node.

Fig. 14 compares energy consumption with different operation modes and communication modules enabled. Subfigures A, C, and E (of Fig. 14) show the effect of buffering when the same transmission module is used. The energy use is the highest in the case without buffering (configuration A). When the pre-transmit buffer is included (scenario B), energy is reduced up to some point, and with the second buffer, the reduction is more significant. Scenario C with LoRaWAN is at an energy usage level of 42.63% compared to scenario A with the same communication module (20122 mA vs 47200 mA). The difference between scenarios A and C with the integrated Wi-Fi module is 21.71% (20237 mA vs 93200 mA). The biggest gain is with GPRS, where the energy needed for scenario C is only 8.36% (20620 mA vs 246400 mA).

When comparing the same operating scenario against different communication modules (Fig. 14 – B, D, and F), the most significant difference is for scenario A. The introduction of a buffer would close the gaps. For scenario C, the power usage with the GPRS module is less than 3% higher than with LoRaWAN.

This result is promising for implementing the nodes running in an off-grid regime. When they operate in near real-time with the most effective configuration (scenario A and with LoRaWAN), the node uses a predictable amount of energy. The battery could last several more days without recharging than the design based only on ESP32. The node must adapt its behavior if the external conditions worsen or the LoRaWAN module stops working correctly. So, it should switch to more energy-consuming communication devices, such as the GPRS. With the consumption estimate, the node could calculate the remaining energy and raise the appropriate alarm. Depending on the battery charging rate, buffering could be turned on, and the message queue size could be adjusted. In this way, the node could reduce energy consumption on the cost of near real-time reporting.

In the cloud system, in the database layer, each IoT node has been represented by the configuration data sequence. These data are sensor addresses, retrieval and retention period, boundary (minimal and maximal), or set of accepted values. The copy of all these data is then moved to the memory of the IoT node connected to specific sensors. In this way, every IoT node is fully aware of all connected sensors and their behavior. In this situation, verifying the sensor or connection line failure is more accessible. The most common conditions are when the IoT receives data from a sensor in an irregular interval, with values out of bounds, or when no response from the sensor can be detected. The response to all the mentioned scenarios could be predefined in the IoT node software, making the system reaction faster and more predictable. Also, the software change, if needed, is a much easier task in IoT than at the sensor network level.

7.1. Comparison with Industrial Standard Solution

Since IoT is an essential element of the Industry 4.0 landscape, many successful solutions are available. During the development process, we designed our solution based on our experience with Cassia [15], Aegex [58], and BARTEC [10], and with special requirements faced in hazardous and remote areas for the device with low build, maintenance, and operational costs (Table 8).

The usual approach for hazardous areas is gateway-centric architecture. This means that the complete system consists of multiple devices, some of which are sensors, some of which are concentration nodes, and some of which are gateways. Such approaches bring robust and very potent solutions, but from an explorational point of view, they are more convenient for more extensive facilities with constant human presence. The gateway-centric approach comes with dedicated on-site supporting hardware. The three IoT systems have their own hardware devices for monitoring and maintenance. Our solution could be monitored by any device with LoRaWAN connectivity, authorized through our cloud, and installed with dedicated software. Another advantage of gateway-centric architecture is the possibility of extending the system over the API, while the presented solution only supports application-level software updates. Our solution has been developed to work in IoT-centric mode, where only one type of node plays a leading role in data collection, aggregation, and transmission processes.

Table 8. Comparison of the main features of similar industrial solutions

Feature	Cassia [15]	Aegex [58]	Bartec [10]	Presented solution
Architecture type	Gateway-centric	Both Gateway- and IoT-centric	Both Gateway- and IoT-centric	IoT-centric
On-site HW support	Cassia IoT Access Controller w Bluetooth PnP	Custom-built intrinsically safe tablet device Wi-Fi connected	Custom-built Android-based smartphone	LoRaWAN complaint device
Level of SW extensibility	Application and API	Application and API	Application and API	Application
Sensor connectivity	Spec. sensors Bluetooth	Spec. PnP sensors LAN, WiFi, Bluetooth	Spec. PnP sensors LAN, WiFi, Bluetooth	Any I2C or RS485 sensor Software level adaptation
Sensors per device	Practically unlimited	8 per gateway 4 per IoT device	Practcally unlimited 1 for HY LOG	4 per device
GSM module	External	Integrated	External Internal(HY LOG)	Integrated
GPS module	External	Integrated	External Internal (HY LOG)	Integrated
Power options	AC or DC; battery backup	AC or DC; battery backup; External solar system	AC or repl. battery; Solar (HY LOG)	Integrated or external solar system

Regarding connectivity and supported sensors, Aegex and BARTEC support manufacturer-specific sensors as separate devices that could be added to a network plug-and-play manner using LAN, Bluetooth, or Wi-Fi. At the same time, Cassia's solution relies only on

Bluetooth for connection. On the other hand, our solution works on a bit lower level, offering I2C and RS485 connectivity for any low-level sensor with such possibility. Our solution allows connecting to 4 sensors, the same as the Aegex solution. Aegex solution would need a gateway for each IoT node, while our solution gateway node is unnecessary.

The most similar solution to our node is BARTEC HY LOG. It is a complete system in one enclosure dedicated to monitoring the quantity of hydrogen. This device also supports GSM connectivity and GPS tracking by default, but it is committed to only one task. Like our IoT node, it has an incorporated solar panel and can run independently from a wired power supply. Other systems support integration with GSM, GPS, and solar-powered battery power supplies, but only through external devices, which makes the system much more extensive and complex for installation.

The proposed solution is a complete system in one device, intended to work without human intervention and with the possibility of connecting to any sensor running supported connection interfaces. It offers software-level flexibility, which means that the nodes with the same hardware configuration in the same network can run different pieces of software and perform different tasks. The basic architecture is IoT-centric, meaning separate gateway devices are unnecessary, and all nodes connect independently to higher levels. The consequence of this approach is that scalability is not dependent on the IoT level and its features and characteristics but on the performance and capacity of the higher-level machines. Since all nodes are equal, maintenance is done by simply replacing the malfunctioned device with a new one and initiating the OTA setup after the new node has been registered in the cloud server.

7.2. Limitations

When building the experimental setup, we considered the deployment in remote locations on the top of the objects containing dangerous fluids. In such cases, the node would be under constant exposure to sunlight (during daytime), and the charging should be close to optimal; thus, the choice to locate a node on the rooftop would be a likely solution for the test location. We must note that the area of South Europe is an environment with very high solar exposure levels. In our location, the solar exposure is rarely under 1.5 kWh/m2, even in the winter months [37], which is higher than the certification exposure rate for solar panels (usually 1 kWh/m2). Compared to colder and cloudier environments, such a setup will provide faster charging and more energy for the battery.

Indoor experiments were mainly focused on building the node configuration and partly on checking the charging in the cases where the node will receive a lower amount of light. Generally, the node could be placed differently from the connected object, leading to partial daily sunlight exposure. For this reason, we conduct the experiments indoors where the node will be in the shade, or even covered, for the day and receive less solar energy. These experiments are part of our work on the improvements in battery charging routine, and they will be presented in future papers. However, this still could not cover all real-world exploitation problems, including many additional issues, such as mechanical damage, node disconnection, and various problems that could appear. However, handling unexpected situations is currently supported through different alarm-based routines. They promptly notify the Edge and cloud levels when node operations get jeopardized.

Another limitation is that besides their widespread usage in IoT implementation, MQTT protocols have particular security vulnerabilities [29]. The most common vulnerability is a connection without encryption, which usually results from choosing the fastest possible unencrypted communication. To suppress the vulnerabilities resulting from communication over unencrypted channels, the proposed IoT node's security is improved using SSL/TLS protocol combined with internally supported ESP32 hardware encryption [20]. We rely on the MQTT-integrated SSL/TLS protocol for transport and payload encryption, using port 8883 by default. This approach is suitable for ESP32-based applications due to its processing power. Combined with ESP32 hardware-level encryption, it ensures data integrity and prevents reverse code engineering. There is room to introduce additional encryption and countermeasures without increasing the amount of energy used.

The general drawback of these approaches is that the node will still be vulnerable to known SSL/TLS and ESP32 crypto-related problems but much safer than the use without any cryptography measure. Also, programmers must carefully write the code to prevent "lapses in developer awareness" [29] by exposing critical pieces of information, and any additional computation will use more energy in the node.

The use of adequate communication channels is the next point to mention. Since the ESP32's integrated Wi-Fi and Bluetooth were not suitable for intended use, they have been replaced by LoRaWAN and GPRS modules. Later, two communication protocols were designed to support long-range communication. Due to their energy consumption levels, LoRaWAN has been chosen as the primary and GPRS as the backup channel. Generally, the main drawbacks of LoRaWAN are low bandwidth and packet size. The low bandwidth itself peaks at around 10kbit/s in Europe [55] and it is unsuitable for high loads of real-time data, thus low packet size. In our testing scenarios, we observed that when collecting the higher amount of data in the transmission queue, they could not be sent within a single transmission. Still, they must be split into several messages.

A similar limitation is for GPRS connection, too. GPRS has a higher sending data rate and more significant message size limitation [52], but it is significantly slower and under the capacity of standard Wi-Fi in these terms, too. Since the primary requirement is to achieve data transmission over long distances, LoRaWAN and GPRS have been chosen as the more adequate means of transport. Communication-based on LoRaWAN is widely used in this device class both in experimental and industrial applications.

The proposed platform runs on the ESP32, which performs excellently compared to earlier versions of microcontrollers used for IoT applications. However, it still does not provide the processing level or memory capacity of the Edge or cloud-level computers. The primary limitation in terms of software complexity, flexibility, and scalability on the node's level lies in the hardware background of the node itself and its use at the IoT level. The situation is different if we look at the whole picture of the system consisting of IoT, Edge, and Cloud level nodes, with the possibility of updating IoT nodes in the OTA manner. Since the node is registered in the cloud by its MAC address, the server at the cloud level could initiate a software update and replace the existing set of routines with another, significantly changing the node's role. In that way, the node's flexibility could be improved. In such a scenario, the limitation would be hardware connection to sensors, if any, which must be replaced to ensure proper node functioning.

7.3. Reliability Analysis and Next Steps

Future work will enhance IoT nodes by employing redundancy and reliability improvement schemes, such as failure partners. In this way, nodes will be able to cover more scenarios that are outside their current niche. Currently, redundancy is supported on a sensor level. A single IoT node can monitor multiple sensor devices of the same type (usually two), and they can act as failure partners. In this scenario, the operation node uses one sensor until its return values are within a predefined range. When the sensor returns unbalanced or out of the predefined range values, the IoT node will raise the alarm and switch to the backup sensor. This complete control is done on the software level. It is worth mentioning that such an approach will result in lower energy consumption but with lower flexibility.

The update of the failure partner scenario at the sensor level will be the approach when both sensors are active simultaneously. In this case, the IoT node compares results, and when one of them starts generating invalid values, the IoT node completely switches to the one that functions correctly. The sensor in a failure state could then be shut down, and an adequate alarm could be generated. When the sensor malfunction gets repaired or replaced, it will send the notification signal to the IoT node, which will start the recovery procedure. This approach does not guarantee 100% reliability since there is always a chance that both sensors could go to the failure state. In this case, the system will react by raising the highest priority alarm. The same type of alarm will also be raised when the sensor gets to an error state, but no redundancy device is installed. When only one sensor is present and it fails, the situation is beyond software-directed recovery, and physical intervention must be done. This update will also be entirely on a software level.

One of the limits is the possibility of replacing the processing and communication modules. They are in the device casing, so any repair or replacement action would require node disconnection and replacement. For this reason, introducing redundant IoT nodes will be one of the possible solutions. Another possibility for improvement would be reconfiguring the complete network by introducing different IoT nodes with different roles. When one would be used only for data collection, the others could be used for data processing and transmission. This way, the system would be more robust and reliable but at a higher maintenance cost since more nodes must be employed and more software variants must be maintained. Such an improvement would move the architecture towards a gateway-centric model, but with all nodes running the same hardware.

The introduction of redundant IoT nodes is the solution to handle cases with hardware errors. In a configuration with two IoT nodes, both have an equal structure and have the same software installed. One of them acts as a master, and the other one is a slave. The configuration with master and slave IoT nodes is a shift away from IoT-centric design since both nodes must be connected to the same set of sensors over the communication line. This would result in more expensive solutions and a significant shift to gateway-centric architecture. Compared to redundant partner design, the difference is that only the master can trigger data exchange with the Edge level. At the same time, the slave will only listen to the traffic and receive the data sent by the sensors. In this situation, the master IoT node is active, and the slave is in the so-called sniffer mode. When the IoT node is in the sniffer node, it sends no data to higher levels (Edge computer).

When the slave node does not receive the keep-alive message for the predefined period, it will try to connect to the master node (ping). If there is no response from the master node, the slave will switch to the active (master) mode. At that moment, the former slave IoT node will take over the complete functionality of the former master and set up all the functions needed for the sensor and Edge layers. This procedure will be executed without human intervention, and when such an incident happens, the new master node will send a high-level alarm to the Edge layer. Also, regarding software updates or hardware replacements, one node could be shut down for updates while the other will continue to collect measurements. Research in this direction would also switch the deployment paradigm to gateway-centric design, bringing higher reliability but at a higher maintenance cost. With such an update, the solution will be more suitable for more extensive deployments and leave its current niche. Expanding communication to higher levels will focus on security. Currently, both ESP32 and additional communication modules support basic 802.11 security standards. Since this could be easily broken, one of the focuses for the next phase will be the acquisition of advanced security protocols for IoT devices.

The presented research was focused on the design of the single node. In terms of scalability, it is equal to the scalability of its building blocks. The most important feature of the design is the possibility of integrating the IoT node into broader systems. The node can communicate with the environment using two channels (LoRaWAN and GSM) and, optionally, two channels that come as part of ESP32 (Wi-Fi and Bluetooth). The proposed IoT nodes could theoretically cover unlimited sensing devices by participating in the more comprehensive network. Each IoT node could connect to RS485 and I2C and transmit data to the Edge level. Using the MQTT-SN protocol, the designed IoT node can connect to every system that supports such communication.

Improvements in the battery charging algorithm would be necessary for future design improvements. As the first step, we introduced externally controlled charging, which could be triggered from the Cloud or Edge level and force the IoT node to start to charge the battery. Next, we replaced simple threshold-based charging with an improved process that considers the current battery level, the estimated energy consumption, and the time until the next sunrise. The focus is currently on defining the method based on the improved techniques and machine learning to define autonomous models, which will ensure, if possible, IoT node operation in the off-grid environment.

8. Conclusion

The paper introduces a novel combination of energy-efficient hardware selection and adaptive software control to manage power consumption autonomously. Multiple limitation factors, such as casing design, cost, and the worldwide availability of used components, drove the design request. The starting point was a solely used ESP32, and during the development, the inefficient hardware elements were replaced, and an autonomous power supply system was integrated. This was a challenge because used components were often designed to run in factory conditions without power or connectivity limitations.

Thanks to the advanced operating system of the ESP32 node, further improvements were made through the set of software implementations and updates, including the definition of the optimized working mode. By integrating hardware and software optimizations, this work improves upon traditional IoT designs for Industry 4.0, offering enhanced efficiency for deployment in remote and hazardous environments. This research was conducted in parallel with investigating diverse deployment strategies for client software across various ISA-95 layers. Throughout this process, the node was integrated into a digital twin structure in the cloud, and the possibility of the software OTA update and monitoring was enabled. Overall, all software design and hardware configuration optimizations aimed to enhance energy efficiency (Table 9), and this goal was achieved by:

- Implementing different battery charging routines to maximize energy collection effectiveness. Since the standard battery charging routine triggers relatively rarely (once a week or bi-weekly), automatic charging could start at night or in bad weather, resulting in no energy gain. To suppress this, a controlled charging mode, initiated from the Edge level, was implemented, which could trigger battery charge on demand, by a predefined schedule, or based on the weather forecast.
- Utilizing external low-power communication components. The LoRaWAN component for real-time transmission reduces energy use by nearly half (50.64
- Defining a new controlled active mode optimized for the anticipated use. The new mode with the communication part disabled utilizes 72% of the energy used in comparable modem sleep mode (36 mA vs. 50 mA) and only 40% of the power that would model sleep mode with active sensors (69 mA vs. 149-200 mA) would use. A similar ratio applies when sensors and the LoRaWAN module are active – 98 mA vs. 200 mA when ESP32 is in standard active mode with sensors enabled.
- Implementing adaptive software that ensures seamless transitions between active and sleep modes. Based on the required measurement, processing, and transmission frequencies, the controlling software will decide when to switch the active components off and reduce energy consumption.
- Integration into digital twin that allows early warning mechanisms and OTA updates. The frequency of transmission of node health parameters to digital twin could be configured, but their size is the equivalent of a single packet containing data collected from sensors. Usually, it is enough to run such a telemetry for once after 1000 data collection cycles. The additional energy consumption caused by such a process would be less than 0.1
- Using message buffers to reduce the number of data transmissions. For the most common scenario with LoRaWAN, using a buffer of size ten will result in an energy reduction of 25%, while using a buffer of size 100 will result in a reduction of up to 55%. When a message buffer of size 100 is used, the total energy consumption will be very close regardless of the transmission module used.

The more notable gain is when GPRS is used for transmission. If a buffer of only ten messages were used, only 22.40% of the initially required energy would be used. In contrast, with a buffer size of 100, the consumption will be reduced to 9.38%. Notably, this approach introduces a trade-off: while it reduces energy usage, reporting to the Edge layer will be less frequent.

Continued improvement efforts are directed toward enhancing system reliability, fault tolerance, information security, and overall system readiness and availability. As a preliminary step, we envision enhancing reliability by introducing additional redundancy at the IoT level, bolstering robustness and error resilience. Further improvements to the battery charging subsystem, as well as possible security updates and vulnerability prevention, will also run in parallel with ongoing node development, aiming to extend battery life and mitigate the risk of power depletion.

An ancillary outcome of this research is a set of design recommendations formulated during the enhancement process:

– *Standardized Components*: Adhere to proven standardized components that have demonstrated reliability in real-world conditions.
– *Module Disabling and Replacement*: Permanently disable or replace modules that fail to meet performance expectations.
– *Feature Flags for Dark Mode*: Introduce feature flags to enable dark mode in regular software operations (not exclusively for software updates).
– *Message Queues and Buffering*: External management of message queues and buffering must be employed to adapt the node's operation dynamically.
– *Integration with Digital Twins*: Enable permanent monitoring by integrating IoT nodes with digital twins.

Table 9. Energy-saving enhancements

Updated	Compared aspect	Energy reduction
CAM Mode	ESP32 Light Sleep Mode	20–30%
CAM Mode	ESP32 Active Mode	45–55%
CAM + Sensors	Sensor reading and ESP32 processing in active mode	50–70%
LoRaWAN	ESP32 integrated Wi-Fi	50%
Transmission buffer of size 100	Immediate transmission upon processing. The used energy is nearly equal regardless of the transmission device	55–90%

The node is designed to fill a niche in the industrial solution landscape opened by the need for an IoT-centric system consisting of an array of nodes based on the same hardware and able to run different pieces of software. Such nodes could form an OTA-controlled network where each element could be easily reconfigured and take on a new role.

This paper focuses on the node's general hardware and software structure and its primary role – collecting, processing, and transmitting data read from the sensors, using the lowest possible amount of energy, and having the complete node delivered in a single packaging.

While the presented node operates within a specific industrial context, the solutions it embodies transcend disciplinary boundaries. Authors must remain receptive to diverse concepts, regardless of their research origins. This study underscores the ongoing need to continually enhance energy-efficient component usage, evaluating and incorporating solutions as they prove sufficient.

Acknowledgments. This work has been supported by the cost action CA 19135 CERCIRAS (Connecting Education and Research Communities for an Innovative Resource Aware Society). This work has been funded by the Ministry of Education, Science, and Technological Development of the Republic of Serbia, grant number 451-03-68/2022-14/ 200102

Nomenclature

Acronym	Description
ACH	Average energy Consumption per Hour
AL	Alarm Low energy level in battery.
CAM	Controlled Active Mode
CL	Charging required Level
ESP32	Low-power microcontrollers are widely used in IoT applications.
Ex e	The class of device enclosure constructed and certified as explosion-protected according to the Increased Safety standard.
FreeRTOS	Free Real Time Operation System. Operation system native to ESP32 controller
GPRS	General Packet Radio Service, data transfer standard for mobile networks
GPS	Global Positioning System. Satellite-based radio navigation system.
GSM	Global System for Mobile communications, standard for mobile networks
I2C	Inter-Integrated Circuit. Serial communication bus used to attach lower speed sensors
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things
ISA-95	Standard from the International Society of Automation for developing an automated interface between enterprise and control systems.
LoRa	Low Radiation. Network protocol to wirelessly connect battery-powered devices.
MAC	Media Access Control
MQTT	Message Queuing Telemetry Transport protocol
MQTT-SN	Message Queuing Telemetry Transport for Sensor Networks protocol
OTA	Over-The-Air. Update to an embedded system that is delivered through a wireless network
RH	Requested High level. Battery level where charging should stop.
RS485	Recommended Standard #485. The standard for serial communication between devices
RTC	Real-Time Clock
SH	Standard High battery level
SIM	Subscriber Identification Module. The card is used to enable mobile communication for devices.
SSL/TLS	Secure Socket Layer / Transport Layer Security
SL	Standard Low Battery Level
ULP	Ultra-Low Power. Processing unit optimized for low energy consumption.
UMTS	Universal Mobile Telecommunication System. Cellular system for network based on GSM

References

1. Aheleroff, S., Xu, X., Lu, Y., Aristizabal, M., Velásquez, J.P., Joa, B., Valencia, Y.: Iot-enabled smart appliances under industry 4.0: A case study. Advanced engineering informatics 43, 101043 (2020)
2. by Akacia System, D.: GDM-8255A Dual Display Digital Multimeter — gwinstek.com. `https://www.gwinstek.com/en-global/products/detail/GDM-8255A`, [Accessed 05-11-2024]
3. Al-Kashoash, H.A., Kemp, A.H.: Comparison of 6lowpan and lpwan for the internet of things. Australian Journal of Electrical and Electronics Engineering 13(4), 268–274 (2016)
4. Aleksic, D.S., Jankovic, D.S., Rajkovic, P.: Product configurators in sme one-of-a-kind production with the dominant variation of the topology in a hybrid manufacturing cloud. The International Journal of Advanced Manufacturing Technology 92, 2145–2167 (2017)
5. Aleksić, D.S., Janković, D.S., Stoimenov, L.V.: A case study on the object-oriented framework for modeling product families with the dominant variation of the topology in the one-of-a-kind production. The International Journal of Advanced Manufacturing Technology 59, 397–412 (2012)
6. Anbazhagan, S., Mugelan, R.: Energy efficiency optimization of nb-iot using integrated proxy & erai technique. Results in Engineering 23, 102419 (2024)
7. Andres-Maldonado, P., Lauridsen, M., Ameigeiras, P., Lopez-Soler, J.M.: Analytical modeling and experimental validation of nb-iot device energy consumption. IEEE Internet of Things Journal 6(3), 5691–5701 (2019)
8. Baig, M.J.A., Iqbal, M.T., Jamil, M., Khan, J.: Design and implementation of an open-source iot and blockchain-based peer-to-peer energy trading platform using esp32-s2, node-red and, mqtt protocol. Energy reports 7, 5733–5746 (2021)
9. Banguero, E., Correcher, A., Pérez-Navarro, Á., Morant, F., Aristizabal, A.: A review on battery charging and discharging control strategies: Application to renewable energy systems. Energies 11(4), 1021 (2018)
10. Bartec: Industrial Internet of Things for hazardous areas: potential for the optimisation of existing plants Whitepaper. `https://bartec.com/fileadmin/2-Products_and_Solutions/2-5-Smart_Factories/ACS_Whitepaper_EN.pdf`, [Accessed 05-11-2024]
11. blog, E.B.: ESP32 Alternatives: Pros, Cons & Best Options in 2023 — espboards.dev. `https://www.espboards.dev/blog/esp32-alternatives`, [Accessed 05-11-2024]
12. Bose, B., Garg, A., Panigrahi, B., Kim, J.: Study on li-ion battery fast charging strategies: Review, challenges and proposed charging framework. Journal of Energy Storage 55, 105507 (2022)
13. Bouguera, T., Diouris, J.F., Chaillout, J.J., Jaouadi, R., Andrieux, G.: Energy consumption model for sensor nodes based on lora and lorawan. Sensors 18(7), 2104 (2018)
14. Brous, P., Janssen, M., Herder, P.: The dual effects of the internet of things (iot): A systematic review of the benefits and risks of iot adoption by organizations. International Journal of Information Management 51, 101952 (2020)
15. Cassia: Industrial IoT Products and Solutions — cassianetworks.com. `https://www.cassianetworks.com/bluetooth-iot-solutions/industrial-iot/`, [Accessed 05-11-2024]
16. Dos Anjos, J.C., Gross, J.L., Matteussi, K.J., González, G.V., Leithardt, V.R., Geyer, C.F.: An algorithm to minimize energy consumption and elapsed time for iot workloads in a hybrid architecture. Sensors 21(9), 2914 (2021)
17. Ensworth, J.F., Reynolds, M.S.: Ble-backscatter: Ultralow-power iot nodes compatible with bluetooth 4.0 low energy (ble) smartphones and tablets. IEEE Transactions on Microwave Theory and Techniques 65(9), 3360–3368 (2017)

18. Espressif: Esp32 series datasheet. `https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf`, [Accessed 05-11-2024]
19. Fowler, M.: bliki: Dark Launching — martinfowler.com. `https://martinfowler.com/bliki/DarkLaunching.html`, [Accessed 05-11-2024]
20. Hyde, J.: Hardware security by design: Esp32 guidance. `https://www.nccgroup.com/us/research-blog/hardware-security-by-design-esp32-guidance` (2022), [Accessed 05-11-2024]
21. IAEA: 14. guidelines for integrated risk assessment and management in large industrial areas. `https://www-pub.iaea.org/MTCD/publications/PDF/te_994_prn.pdf`, [Accessed 05-11-2024]
22. ISA: ISA95, Enterprise-Control System Integration- ISA — isa.org. `https://www.isa.org/standards-and-publications/isa-standards/isa-standards-committees/isa95`, [Accessed 05-11-2024]
23. Jeon, K.E., She, J., Xue, J., Kim, S.H., Park, S.: luxbeacon—a batteryless beacon for green iot: Design, modeling, and field tests. IEEE Internet of Things Journal 6(3), 5001–5012 (2019)
24. Jia, K., Xiao, J., Fan, S., He, G.: A mqtt/mqtt-sn-based user energy management system for automated residential demand response: Formal verification and cyber-physical performance evaluation. Applied Sciences 8(7), 1035 (2018)
25. Kanan, R., Elhassan, O., Bensalem, R.: An iot-based autonomous system for workers' safety in construction sites with real-time alarming, monitoring, and positioning strategies. Automation in Construction 88, 73–86 (2018)
26. Khutsoane, O., Isong, B., Gasela, N., Abu-Mahfouz, A.M.: Watergrid-sense: A lora-based sensor node for industrial iot applications. IEEE Sensors Journal 20(5), 2722–2729 (2019)
27. Kumar, K., Chaudhri, S.N., Rajput, N.S., Shvetsov, A.V., Sahal, R., Alsamhi, S.H.: An iot-enabled e-nose for remote detection and monitoring of airborne pollution hazards using lora network protocol. Sensors 23(10), 4885 (2023)
28. Kumar, S., Tiwari, P., Zymbler, M.: Internet of things is a revolutionary approach for future technology enhancement: a review. Journal of Big data 6(1), 1–21 (2019)
29. Lakshminarayana, S., Praseed, A., Thilagam, P.S.: Securing the iot application layer from an mqtt protocol perspective: Challenges and research prospects. IEEE Communications Surveys & Tutorials (2024)
30. Mcginthy, J.M., Michaels, A.J.: Secure industrial internet of things critical infrastructure node design. IEEE Internet of Things Journal 6(5), 8021–8037 (2019)
31. Mocnej, J., Miškuf, M., Papcun, P., Zolotová, I.: Impact of edge computing paradigm on energy consumption in iot. IFAC-PapersOnLine 51(6), 162–167 (2018)
32. Monteil, T.: Integration of green aspect inside internet of things standard. In: 2023 Congress in Computer Science, Computer Engineering, & Applied Computing (CSCE). pp. 1753–1760. IEEE (2023)
33. Muralidhar, T.V., Sandeep, V.V.S., Manohar, P., Krishna, M.L., Ruthvik, K., Bagwari, S.: An iot based real time forest fire detection & alerting system using lora communication. In: 2024 11th International Conference on Signal Processing and Integrated Networks (SPIN). pp. 139–144. IEEE (2024)
34. NSW.GOV.AU: Increased safety Ex e — nsw.gov.au. `https://www.nsw.gov.au/testsafe/electrical/explosive-atmosphere/increased-safety`, [Accessed 05-11-2024]
35. Paiola, M., Gebauer, H.: Internet of things technologies, digital servitization and business model innovation in btob manufacturing firms. Industrial Marketing Management 89, 245–264 (2020)
36. Phuyal, S., Bista, D., Bista, R.: Challenges, opportunities and future directions of smart manufacturing: a state of art review. Sustainable Futures 2, 100023 (2020)

37. Potić, I., Golić, R., Joksimović, T.: Analysis of insolation potential of knjaževac municipality (serbia) using multi-criteria approach. Renewable and Sustainable Energy Reviews 56, 235–245 (2016)
38. Qu, Y., Ming, X., Liu, Z., Zhang, X., Hou, Z.: Smart manufacturing systems: state of the art and future trends. The International Journal of Advanced Manufacturing Technology 103, 3751–3768 (2019)
39. Rajab, H., Al-Amaireh, H., Bouguera, T., Cinkler, T.: Evaluation of energy consumption of lpwan technologies. EURASIP Journal on Wireless Communications and Networking 2023(1), 118 (2023)
40. Rajković, P., Aleksić, D., Djordjević, A., Janković, D.: Hybrid software deployment strategy for complex industrial systems. Electronics 11(14), 2186 (2022)
41. Rajković, P., Aleksić, D., Djordjević, A., Janković, D.: Hybrid software deployment strategy for complex industrial systems. Electronics 11(14), 2186 (2022)
42. Rajković, P., Aleksić, D., Janković, D.: The implementation of battery charging strategy for iot nodes. In: European Conference on Parallel Processing. pp. 40–51. Springer (2023)
43. Rajković, P., Djordjević, A., Aleksić, D., Janković, D.: Usage of modular software development for iot nodes— a case study. In: Proceedings of the Tenth Workshop on Software Quality Analysis, Monitoring, Improvement, and Applications SQAMIA. pp. 114–125. Bratislava, Slovakia (2023), https://ceur-ws.org/Vol-3588/p11.pdf
44. Roldán-Gómez, J., Carrillo-Mondéjar, J., Castelo Gómez, J.M., Ruiz-Villafranca, S.: Security analysis of the mqtt-sn protocol for the internet of things. Applied Sciences 12(21), 10991 (2022)
45. RTOS, F.: FreeRTOS™ - FreeRTOS™ — freertos.org. `https://www.freertos.org`, [Accessed 05-11-2024]
46. Schaarschmidt, M., Uelschen, M., Pulvermüller, E.: Hunting energy bugs in embedded systems: A software-model-in-the-loop approach. Electronics 11(13), 1937 (2022)
47. Semtech: SX1268 — semtech.com. `https://www.semtech.com/products/wireless-rf/lora-connect/sx1268`, [Accessed 05-11-2024]
48. Shekarisaz, M., Kargahi, M., Thiele, L.: Inter-task energy-hotspot elimination in fixed-priority real-time embedded systems. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (2024)
49. Shekarisaz, M., Thiele, L., Kargahi, M.: Automatic energy-hotspot detection and elimination in real-time deeply embedded systems. In: 2021 IEEE Real-Time Systems Symposium (RTSS). pp. 97–109. IEEE (2021)
50. SHOP, C.: Esp32-wroom-32 datasheet. `https://cdn-shop.adafruit.com/product-files/3320/3320_module_datasheet.pdf`, [Accessed 05-11-2024]
51. SIMCom: SIM800H datasheet, design equivalent, SIM Com — datasheetspdf.com. `https://datasheetspdf.com/pdf/823439/SIMCom/SIM800H/1`, [Accessed 05-11-2024]
52. spiceworks: GPRS Working, Advantages, Applications — spiceworks.com. `https://www.spiceworks.com/tech/networking/articles/what-is-gprs/`, [Accessed 05-11-2024]
53. Stanford-Clark, A., Truong, H.L.: Mqtt for sensor networks (mqtt-sn) protocol specification. International business machines (IBM) Corporation version 1(2), 1–28 (2013)
54. Sundaram, J.P.S., Du, W., Zhao, Z.: A survey on lora networking: Research problems, current solutions, and open issues. IEEE Communications Surveys & Tutorials 22(1), 371–388 (2019)
55. thethingsnetwork: LoRaWAN Limitations, The Things Network. `https://www.thethingsnetwork.org/docs/lorawan/limitations/,`, [Accessed 05-11-2024]
56. Uelschen, M., Schaarschmidt, M.: Software design of energy-aware peripheral control for sustainable internet-of-things devices. In: Proceedings of the 55th Hawaii International Conference on System Sciences. Maui, HI, USA (2022)

57. Uni-Trend: UT71 Series Intelligent Digital Multimeters - UNI-T Meters — Test & Measurement Tools and Solutions — meters.uni-trend.com. `https://meters.uni-trend.com/product/ut71-series`, [Accessed 05-11-2024]
58. Ventulett, T.: Aegex iot platform for hazardous locations. `https://aegex.com/images/uploads/Aegex_IoT_Platform_For_Hazardous_Locations_FINAL-1.pdf`, [Accessed 05-11-2024]
59. Wolf, S., Olarte, J.: Battery market segmentation. Emerging Battery Technologies to Boost the Clean Energy Transition p. 85 (2024)

Petar Rajković is an Assistant Professor at the University of Niš, Faculty of Electronic Engineering. He obtained his Ph.D. in software engineering from the same university and teaches various courses at all levels of study. He is focused on model-driven development and information system research, with practical experience in developing innovative software solutions for industrial automation and public health.

Milan Vesković is an Assistant Professor at the Faculty of Technical Sciences Čačak of the University of Kragujevac. He obtained his Ph.D. in electronics from the Faculty of Technical Sciences Čačak at the same university. His research interests include the application of electronics in IoT, environmental protection, and knowledge representation.

Dejan Aleksić is an Associate Professor at the Faculty of Sciences and Mathematics of the University of Nis, Serbia. He obtained his Ph.D. in software engineering from the Faculty of Electrical Engineering at the same university. His research interests include Product configuration, Mass Customization, One-of-a-kind production, and Industrial IoT.

Dragan S. Janković received a B.Sc., M.Sc., and a Ph.D. in computer science from the Faculty of Electronic Engineering, University of Niš, Serbia, in 1991, 1995, and 2001, respectively. He is a full professor at the Department of Computer Science, Faculty of Electronic Engineering, and head of the Laboratory for Medical Informatics. His research interests include logic design, software development, algorithms, medical informatics, artificial intelligence in medicine, and blockchain technology. He was a participant or project leader in more than 30 research and development projects. He published over 350 scientific papers and 10 technical solutions.

Received: August 07, 2024; Accepted: November 04, 2024.

Computer Science and Information Systems 22(2):533–561 https://doi.org/10.2298/CSIS240722021T

Demystifying Power and Performance Variations in GPU Systems through Microarchitectural Analysis ⋆

Burak Topcu⋆⋆[1], Deniz Karabacak[2], and Işıl Öz[3]

[1] The Pennsylvania State University
Department of Computer Science and Engineering, State College, PA, USA
topcuuburak@gmail.com
[2] Izmir Institute of Technology
Electrical and Electronics Engineering Department, Izmir, Turkey
denizkarabacak@std.iyte.edu.tr
[3] Izmir Institute of Technology
Computer Engineering Department, Izmir, Turkey
isiloz@iyte.edu.tr

Abstract. Graphics Processing Units (GPUs) serve efficient parallel execution for general-purpose computations at high-performance computing and embedded systems. While performance concerns guide the main optimization efforts, power issues become significant for energy-efficient and sustainable GPU executions. Profilers and simulators report statistics about the target execution; however, they either present only performance metrics in a coarse kernel function level or lack visualization support that can enable microarchitectural performance analysis or performance-power consumption comparison. Evaluating runtime performance and power consumption dynamically across GPU components enables a comprehensive tradeoff analysis for GPU architects and software developers. In this work, we present a novel memory performance and power monitoring tool for GPU programs, GPPRMon, which performs a systematic metric collection and provides useful visualization views to guide power and performance analysis for target executions. Our simulation-based framework dynamically gathers SM and memory-related microarchitectural metrics by monitoring individual instructions and reports dynamic performance and power values. Our interface presents spatial and temporal views of the execution. While the first demonstrates the performance and power metrics across GPU memory components, the latter shows the corresponding information at the instruction granularity in a timeline. We demonstrate performance and power analysis for memory-bound graph applications and resource-critical embedded programs from GPU benchmark suites. Our case studies reveal potential usages of our tool in memory-bound kernel identification, performance bottleneck analysis of a memory-intensive workload, performance-power evaluation of an embedded application, and the impact of input size on the memory structures of an embedded system.

Keywords: GPU Computing, Performance monitoring, Power consumption

⋆ This article is an extended version of our previously published workshop paper, ”GPPRMon: GPU Runtime Memory Performance and Power Monitoring Tool, Burak Topçu, Işıl Öz, Workshop on Resource AWareness of Systems and Society (RAW), co-located with International European Conference on Parallel and Distributed Computing (Euro-Par), 2023.”

⋆⋆ Work was done when the author was at Izmir Institute of Technology.

1. Introduction

As high-performance and energy-efficient computation requirements increase in data processing tasks, GPU architectures, with heterogeneous components, play an essential role in accelerating parallel workloads [40, 47]. Since GPU devices have evolved into more complex systems with recent technological developments, efficient execution requires more detailed research effort. As a result of high computational capacity, energy and power issues have become crucial in GPU-based systems [22].

While GPU devices have large computational power with multiple processing units, the performance and energy efficiency may decline for memory-intensive workloads due to high pressure on memory units by concurrently executing multiple threads. While several works discuss the impacts of the memory wall problem for the GPUs [10, 12], there are also studies explaining the memory bottleneck reasons for GPU applications [20, 32] and proposing various improvements for the problem [13, 16, 46, 50]. Additionally, the researchers propose energy-efficient methods for GPU programs [6, 36]. While both performance and energy improvements contribute to the efficient execution of GPU programs, they may compete with each other, and the design decisions become critical and get complicated, requiring the evaluation of the tradeoffs between performance and energy efficiency [4, 9, 17, 42].

Performance and energy efficiency analysis for GPU execution requires low-level measurements at runtime and detailed evaluations of the performance bottlenecks and power consumption. Evaluating the execution for performance and energy efficiency at the kernel function level hides most of the clear evidence for conducting a baseline analysis. We need to perform more fine-grain analysis, where the individual warp instructions are tracked throughout the execution on SM resources. However, profiling and simulation tools collect and report GPU performance results at the kernel level. For instance, the NVIDIA Nsight Compute Tool [27], which presents occupancy, IPC, and memory utilization metrics, operates on a kernel basis. Similarly, the state-of-the-art GPU simulation tools [15, 45] report the performance and hardware metrics for each kernel function. None of the tools directly reports GPU programs' dynamic performance, memory access behavior, and power consumption at runtime. While profiling tools and simulation frameworks report runtime statistics, the software developers and researchers spend additional effort to collect related metrics from the experiments. In other words, several in-house target-specific works still exist to monitor the runtime behavior of a GPU execution, and repetitive studies cause redundant effort.

While the profilers help software developers to understand performance and power consumption information about GPU execution, microarchitecture-level simulators are quite significant in modeling hardware and monitoring the runtime application behaviors. The simulators target micro and macro scales by collecting performance and energy metrics. In GPU-related research, GPGPU-Sim [15] and Multi2Sim [45] simulators have been prominent in offering accurate hardware models for NVIDIA and AMD GPUs, respectively, among the other simulators [3, 7, 34]. Moreover, the developer communities of these simulators have provided continuity by incorporating new GPU architectures and optimizations introduced in GPU hardware and software. Among the top five hundred computer systems [41], 179 use NVIDIA-based GPUs as the accelerator/co-processor technology; specifically, 84 include NVIDIA Volta, and 64 use NVIDIA Ampere devices.

In this work, we build and implement, **GPPRMon**, a performance and power monitoring tool, for GPU programs executing on top of a simulation environment. We aim to close the gap between the profilers' high-level static results and the cycle-accurate simulators' large-volume raw data about the execution. Not only do we set up GPPRMon built on configurable simulation execution, but we also generate abstractions and provide visualization views that are easy to capture and comprehend large amounts of data. Our tool presents both dynamic and configurable simulation execution, and rich architectural profiling views by combining the best of both worlds.

As potential users of the GPPRMon, we target program developers, system architects, and researchers, who are working to optimize GPU software or hardware, considering both performance improvement and energy efficiency. Our simulation-based framework dynamically collects microarchitectural metrics by monitoring individual instructions' issues/completions and reports performance and power consumption information at runtime. Hence, it enables the users to analyze the dynamic behavior of memory accesses and thread blocks during program execution. Based on the detailed characteristics collected at runtime, our visualization interface presents both spatial and temporal views of the execution, where the first demonstrates the performance and power metrics for each hardware component, including memory units and SMs; and the latter shows the corresponding information at the instruction granularity in a cycle-based timeline. Our tool enables the users to perform a fine-granularity evaluation of the target execution by observing instruction-level microarchitectural features related to performance and power consumption at runtime. In this article, we extend our previously published workshop paper [44] by including additional case studies that demonstrate the usage scenarios of our tool. To the best of our knowledge, this is the first work monitoring a GPU kernel's performance by visualizing the execution of instructions for multiple user-configurable scenarios, relating memory hierarchy utilization with performance, and tracking power dissipation at runtime. Our main contributions are as follows:

- We design a systematic microarchitectural metric collection methodology that keeps track of instruction-per-cycle (IPC) per streaming multiprocessor (SM) as a performance metric, instruction execution records for each warp to observe issue and completion cycles, memory statistics per each component in the memory hierarchy to understand the possible impacts on performance and power-related statistics per each GPU component at runtime. We build our configurable collection framework on top of the GPGPU-Sim simulation environment [15], which provides cycle-accurate information about the execution.
- Based on the information gathered by our metric collection module, we design and build a visualization framework that executes in parallel to our collection module and generates displays and charts for each kernel execution with the following three perspectives:
    1. *General View* displays the average IPC among SMs, access statistics of L1 and L2 caches, row buffer utilization of DRAM partitions, and dissipated power by the main components for any execution cycle interval.
    2. *Temporal View* shows the details of the instructions with issue and completion cycles for each thread block at warp level. In addition to power consumption statistics for the sub-components in an SM, we include L1 Data (L1D) cache access

statistics, which are local for each SM, to relate the thread block's performance in the same execution interval.

3. *Spatial View* demonstrates the access information for each on-chip L1D cache, L2 cache in each sub-partition, and row buffers of DRAM banks in each memory partition. Additionally, it shows the power consumption distribution among the memory components in the execution interval.

– We present case studies to demonstrate the potential usages of our framework and its visualizations by performing experiments for memory-bound graph workloads and resource-critical embedded applications. Our tool enables us to perform detailed performance and power analysis for the target GPU executions.

The remainder of this paper is organized as follows: Section 2 presents some background on GPU architecture, CUDA programming model, and the simulator. We explain our design and implementation details for tool construction in Section 3. Then, we present case studies in Section 4, demonstrating usage scenarios of the GPPRMon. Section 5 presents the existent performance and power evaluation studies for GPGPU applications. Finally, Section 6 summarizes the work with some conclusive remarks.

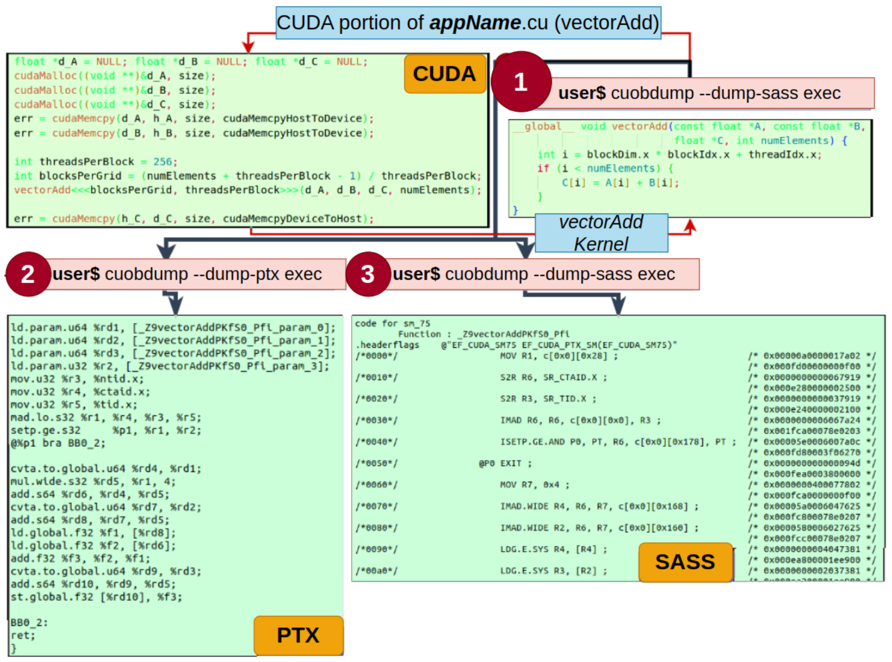
2. Background

2.1. GPU Hardware

Modern GPU architectures employ a single instruction multiple thread (SIMT) execution in the Streaming Multiprocessor (SM) units. Each SM includes multiple warp schedulers, instruction dispatchers, a register file, multiple single and double precision ALUs, tensor cores, special function units (SFU), and load/store units with on-chip (on-SM) local memory. An interconnection network connects SMs to off-chip memory partitions on which DRAM and Last-Level caches (LLC) are placed. While the cores inside the same SM can access the private L1 cache, all the cores can communicate via the L2 cache structure. Load/store instructions may require off-chip accesses whenever requested data cannot be found in the L1D cache. Furthermore, data access gets slower for memory instructions as moving down the hierarchy. GPU architectures have been evolving, with each new generation introducing enhancements in performance, power efficiency, and specialized features. This overview provides a general understanding of GPU architecture, but specific details and terminology may vary based on the GPU device model and manufacturer. We specify the architectural and resource specifications for the GPUs in our experimental setup parts.

2.2. CUDA Programming Model

Compute Unified Device Architecture (CUDA) [26] is an API to execute a function, namely kernel function, on GPUs. A GPU kernel consists of a 3D grid space, where each grid has a 3D thread block with multiple threads. Each sequential 32-thread set forms a *warp* within a thread block in CUDA. When a kernel function launches, the Gigathread engine (thread block scheduler) schedules thread blocks to SMs in the Round-Robin fashion. Register resources on SMs determine the number of active thread blocks, some of which may be issued to the same SMs. Figure 1 demonstrates a GPU kernel code,


CUDA portion of ***appName***.cu (vectorAdd)
CUDA
1
user$ cuobdump --dump-sass exec
vectorAdd Kernel
2
user$ cuobdump --dump-ptx exec
3
user$ cuobdump --dump-sass exec
PTX
SASS

```
float *d_A = NULL; float *d_B = NULL; float *d_C = NULL;
cudaMalloc((void **)&d_A, size);
cudaMalloc((void **)&d_B, size);
cudaMalloc((void **)&d_C, size);
err = cudaMemcpy(d_A, h_A, size, cudaMemcpyHostToDevice);
err = cudaMemcpy(d_B, h_B, size, cudaMemcpyHostToDevice);

int threadsPerBlock = 256;
int blocksPerGrid = (numElements + threadsPerBlock - 1) / threadsPerBlock;
vectorAdd<<<blocksPerGrid, threadsPerBlock>>>(d_A, d_B, d_C, numElements);

err = cudaMemcpy(h_C, d_C, size, cudaMemcpyDeviceToHost);
```

```
__global__ void vectorAdd(const float *A, const float *B,
                          float *C, int numElements) {
    int i = blockDim.x * blockIdx.x + threadIdx.x;
    if (i < numElements) {
        C[i] = A[i] + B[i];
    }
}
```

```
ld.param.u64 %rd1, [_Z9vectorAddPKfS0_Pfi_param_0];
ld.param.u64 %rd2, [_Z9vectorAddPKfS0_Pfi_param_1];
ld.param.u64 %rd3, [_Z9vectorAddPKfS0_Pfi_param_2];
ld.param.u32 %r2, [_Z9vectorAddPKfS0_Pfi_param_3];
mov.u32 %r3, %ntid.x;
mov.u32 %r4, %ctaid.x;
mov.u32 %r5, %tid.x;
mad.lo.s32 %r1, %r4, %r3, %r5;
setp.ge.s32     %p1, %r1, %r2;
@%p1 bra BB0_2;

cvta.to.global.u64 %rd4, %rd1;
mul.wide.s32 %rd5, %r1, 4;
add.s64 %rd6, %rd4, %rd5;
cvta.to.global.u64 %rd7, %rd2;
add.s64 %rd8, %rd7, %rd5;
ld.global.f32 %f1, [%rd8];
ld.global.f32 %f2, [%rd6];
add.f32 %f3, %f2, %f1;
cvta.to.global.u64 %rd9, %rd3;
add.s64 %rd10, %rd9, %rd5;
st.global.f32 [%rd10], %f3;

BB0_2:
ret;
}
```

```
code for sm_75
        Function : _Z9vectorAddPKfS0_Pfi
.headerflags    @"EF_CUDA_SM75 EF_CUDA_PTX_SM(EF_CUDA_SM75)"
/*0000*/                   MOV R1, c[0x0][0x28] ;                         /* 0x00000a0000017a02 */
                                                                          /* 0x000fd00000000f00 */
/*0010*/                   S2R R6, SR_CTAID.X ;                           /* 0x0000000000067919 */
                                                                          /* 0x000e280000002500 */
/*0020*/                   S2R R3, SR_TID.X ;                             /* 0x0000000000037919 */
                                                                          /* 0x000e240000002100 */
/*0030*/                   IMAD R6, R6, c[0x0][0x0], R3 ;                 /* 0x0000000006067a24 */
                                                                          /* 0x001fca00078e0203 */
/*0040*/                   ISETP.GE.AND P0, PT, R6, c[0x0][0x178], PT ;   /* 0x00005e0006007a0c */
                                                                          /* 0x000fd80003f06270 */
/*0050*/               @P0 EXIT ;                                         /* 0x000000000000094d */
                                                                          /* 0x000fea0003800000 */
/*0060*/                   MOV R7, 0x4 ;                                  /* 0x0000000400077802 */
                                                                          /* 0x000fca0000000f00 */
/*0070*/                   IMAD.WIDE R4, R6, R7, c[0x0][0x168] ;          /* 0x00005a0006047625 */
                                                                          /* 0x000fc800078e0207 */
/*0080*/                   IMAD.WIDE R2, R6, R7, c[0x0][0x160] ;          /* 0x0000580006027625 */
                                                                          /* 0x000fcc00078e0207 */
/*0090*/                   LDG.E.SYS R4, [R4] ;                           /* 0x0000000004047381 */
                                                                          /* 0x000ea800001ee900 */
/*00a0*/                   LDG.E.SYS R3, [R2] ;                           /* 0x0000000002037381 */
```

Fig. 1. CUDA, PTX, and SASS code snippets for *vectorAdd* code

vectorAdd, for the vector addition operation. Part $\boxed{1}$ presents a compilation command by nvcc [29], which is the CUDA compiler to generate the executable. Part $\boxed{2}$ demonstrates the Parallel Thread Execution (PTX) [30] instructions, which represent a virtual machine ISA generated by *nvcc*, and Part $\boxed{3}$ includes the SASS machine instructions, which represent low-level machine assembly that compiles to binary code executing on NVIDIA GPU hardware.

2.3. GPGPU-Sim Simulation Framework

A cycle-level microarchitectural simulator GPGPU-Sim [15] (hereafter referred to as the simulator) has been heavily utilized by researchers working on GPU software and hardware optimizations. Figure 2 displays the workflow of the simulator, which configures a traditional NVIDIA GPU and simulates CUDA-based applications. The simulator provides functional and performance-driven modes such that functional mode enables developers to check the kernel's functional correctness, while the performance mode simulates the kernel for the configured GPU in a cycle-accurate manner. The simulator officially supports Volta-based Titan V, V100, RTX2060 GPU series, Pascal-based Titan X, Kepler-based TITAN, and Fermi-based GTX480. Additionally, AccelWattch developers introduced Volta-based RTX2060 S and Ampere-based RTX3070 GPUs to the official simulator configurations. Beyond these, one can reconfigure any GPU with different resources on top of these architecture models. Additionally, the simulator reports achieving 15% performance accuracy error rate with its virtual ISA implementation [15].

The AccelWattch [14], a power model extension of the simulator, is an analytical model formulating power dissipation and utilizing validated power coefficients of mi-

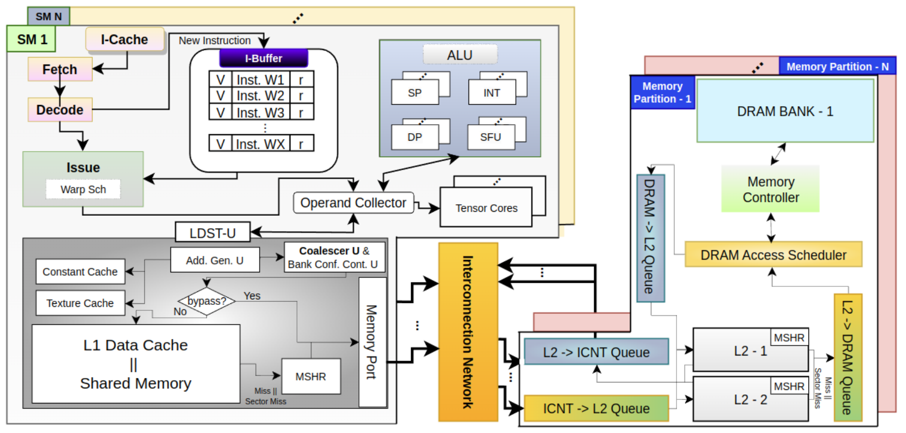


Fig. 2. A general workflow simulation model for a modern GPU

croarchitectural components such as functional units, CUDA core lanes, and memory components, which are gathered through a comprehensive set of experiments. AccelWattch, which supports Dynamic Voltage-Frequency Scaling (DVFS), estimates the energy consumption for V100 with 90% accuracy. In addition to V100, AccelWattch is validated for TITAN X and Turing RTX GPU series through a large set of applications from Rodinia, Parboil, CUTLASS, and DeepBench benchmark suites and NVIDIA CUDA Samples, enables tracking detailed power dissipation of any GPU kernel execution.

Since we build our GPPRMon tool completely on top of GPGPU-Sim and its AccelWattch power model, its accuracy, architectural support and scalability limitations can be considered parallel to the support of the simulation framework.

3. Methodology

GPPRMon tool enables monitoring and visualizing runtime GPU execution performance and power consumption. Figure 3 displays GPPRMon workflow consisting of two main parts: $\boxed{1}$ Metric Collection, $\boxed{2}$ Visualization. For any execution interval, GPPRMon systematically calculates IPC rates of SMs, records warp instruction's issues and completions, collects memory access statistics across the memory hierarchy, and tracks dissipated power on sub-hardware units. Parts $\boxed{1\text{-a}}$ and $\boxed{1\text{-b}}$ demonstrate examples of the power and performance metrics, respectively, and Section 3.1 details what these metrics are and configuration options for users. Furthermore, Part $\boxed{2}$ reveals GPPRMon's visualizer that contains three views to show general performance, memory access statistics, instruction monitoring, and their power dissipation by processing the collected metrics. We build our framework on top of the simulator, which is compatible with many GPU models mentioned in Section 2.3. The GPPRMon framework is available as open-source software [4].

[4] https://github.com/parsiyte/GPPRMon

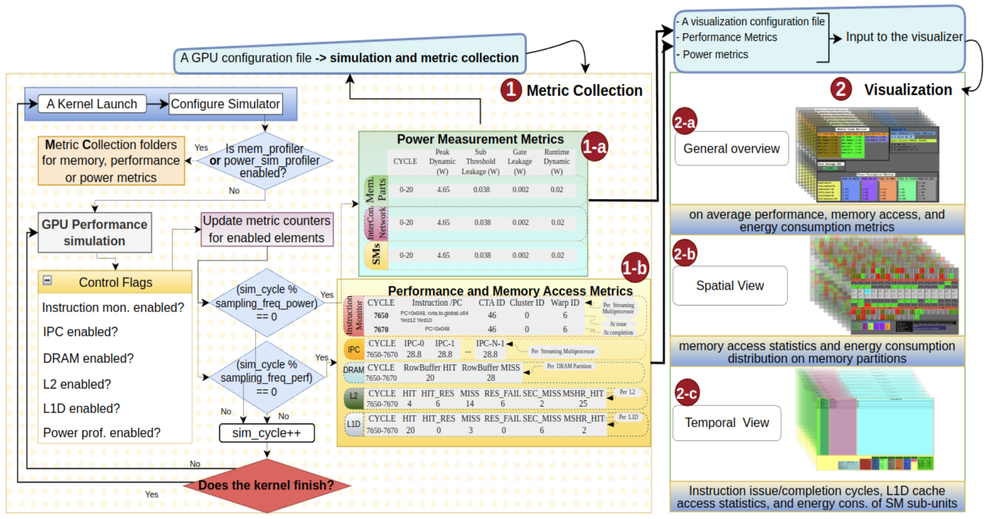

A GPU configuration file -> simulation and metric collection
1 Metric Collection
A Kernel Launch
Configure Simulator
Metric Collection folders for memory, performance or power metrics
Yes
Is mem_profiler or power_sim_profiler enabled?
No
GPU Performance simulation
Update metric counters for enabled elements
Control Flags
Instruction mon. enabled?
IPC enabled?
DRAM enabled?
L2 enabled?
L1D enabled?
Power prof. enabled?
(sim_cycle % sampling_freq_power) == 0
(sim_cycle % sampling_freq_perf) == 0
sim_cycle++
Does the kernel finish?
Power Measurement Metrics
1-a
Mem. Parts
InterCon. Network
SMs
1-b
Performance and Memory Access Metrics
Instruction Monitor
IPC
DRAM
L2
L1D
- A visualization configuration file
- Performance Metrics
- Power metrics
Input to the visualizer
2 Visualization
2-a
General overview
on average performance, memory access, and energy consumption metrics
2-b
Spatial View
memory access statistics and energy consumption distribution on memory partitions
2-c
Temporal View
Instruction issue/completion cycles, L1D cache access statistics, and energy cons. of SM sub-units

Fig. 3. A general workflow overview of **GPPRMon** framework

3.1. Metric Collection

The *Metric Collection* phase of GPPRMon, shown in Part $\boxed{1}$ in Figure 3, conducts the systematical recording of performance and power metrics during the execution. Performance metrics mainly consist of hardware utilization and execution statistics of the memory hierarchy and SMs, such as cache usage efficiency and SMs IPC values. The metric collector extension to the simulator cumulatively tracks the hardware performance, execution statistics, and power counters during a sampling interval, which is determined by *sampling_freq_perf* in Figure 3. At the end of each observation interval, the metric collector exports the tracked metrics to their respective files and clears counters. To enable the performance metric collection feature, the user needs to specify the *mem_profiler* flag in the simulation configuration file. Similarly, power metrics, such as dynamic runtime and peak power, can be collected by setting the *power_sim_profiler* flag.

GPPRMon's metric collection is multi-functional, such that users can separately track warp instruction issue/completion for each thread block, runtime IPC values of each SM, and runtime memory hierarchy utilization statistics (i.e., L1, L2, and DRAM row buffers). GPPRMon allows users to either accumulate or independently collect metrics for each observation interval. Furthermore, users can discard store operations from the memory hierarchy utilization metrics for their runtime target observation. Users can configure these features through metric collection specifications, as depicted *Control Flags* in Figure 3. GPPRMon creates separate folders for each component's statistics and distinct files for each sub-component with IDs. To reduce storage and access overheads during the kernel's execution, we utilize CSV file format for recording metrics. Our visualizer, shown in Part $\boxed{2}$, can execute parallel to Part $\boxed{1}$ and processes the collected metrics to generate runtime visualizations. The following subsections will briefly describe the collected microarchitectural performance and power metrics and how to configure each metric collection separately. More detailed information about how to build GPPRMon and deploy multi-functional metric collection is available on the tool's GitHub page.

Performance Metrics

L1 Data and L2 Caches. A memory request's access status on caches can be one of the possible states: i) Hit: Data resides on the cache line; ii) Hit Reserved: The data is requested, and the corresponding cache line is allocated, but the data is still invalid; iii) Miss: The corresponding cache line causes several sector misses, resulting in a line eviction; iv) Reservation Failure: The situations in which a line allocation, MSHR entry allocation, or merging an entry with an existing in MSHR fail, or the miss queue does not have any slot to hold new requests result in reservation failure; v) Sector Miss: A memory request cannot find the data in the sector of a cache line (i.e., a sector is 32B, whereas a cache line is 128B); vi) MSHR hit: When the upcoming request's sector miss has already been recorded, and request can merge with the existing entry, MSHR hit occurs.

GPPRMon can observe runtime access statistics of each L1D and L2 cache separately. Users can activate the metric collection feature of GPPRMon for L1D and L2 caches with regarding *control flags* as depicted in Figure 3. Tracking runtime cache utilization helps researchers evaluate the application's memory access behavior and relate it to the overall application performance. While cache hits indicate small access latencies, misses generally refer to longer latencies and increasing active memory traffic in the hierarchy. Furthermore, reservation failures result in a memory pipeline stall, directly delaying the subsequent load/store operations. Handling intensive misses requires detailed analytic observations to deduct behavioral interpretations across applications and cache utilization. In this manner, GPPRMon can meet the runtime analytic observation necessity for micro-architectures on GPU to identify the memory performance and power issues understandably.

Row Buffers of DRAM Banks. A row buffer hit occurs when an L2 miss request finds the requested data in the row buffer of the target DRAM bank. A row buffer miss results in a longer access service time than those hits because handling a row buffer miss requires scheduling a memory request to the correct address on DRAM and activation latencies. Row buffer utilization with cache access statistics completes the runtime memory hierarchy behavior exploitation, crucial for describing overall memory performance, especially for sparse data applications. GPPRMon provides separate metric collections for the row buffers in each DRAM partition, and users can activate this feature by enabling *DRAM control flag* as depicted in Figure 3.

SM IPC. An IPC rate mainly describes an SM's performance, which consists of various functional units with varying lane depths. For example, V100 SMs include four single-precision (SP) ALUs with four pipe depths, as presented in Figure 2. When a thread block occupies only SP-ALU lanes, assuming an operation takes one cycle, the ideal IPC for each SM should be sixteen without other functional units' contribution. However, IPC oscillates during the execution depending on SM and memory hierarchy utilization. GPPRMon tracks the runtime IPC rate for each SM separately for each sampling interval, and one can active per SM IPC tracking by enabling *IPC control flag* as in Figure 3. With GPPRMon, developers can analyze runtime IPC variations and investigate the root causes of IPC suffering.

Instruction Monitor. The instruction monitor feature of GPPRMon records the issue/completion cycles of warp instructions within each thread block together with their opcode, operand, and PC separately. Since GPUs execute instructions for multiple threads concurrently by a common PC within a warp, we design tracking instruction issues/completions at the warp level. While the first row of the *Instruction Monitor* in Part $\boxed{\text{1-b}}$ shows the issue statistics, the second displays the completion. Even if the dispatcher unit may issue the same instruction multiple times for a warp, it is guaranteed that any two instructions, of which *CTA_ID*, *SM_ID*, *local Warp_ID*, and *PC* are the same, cannot change the issue/completion sequence. Hence, we can obtain the correct issue/completion matching for each instruction. Users can activate the instruction monitoring utility for each thread block with *instruction mon. control flag* as in Figure 3.

Power Metrics

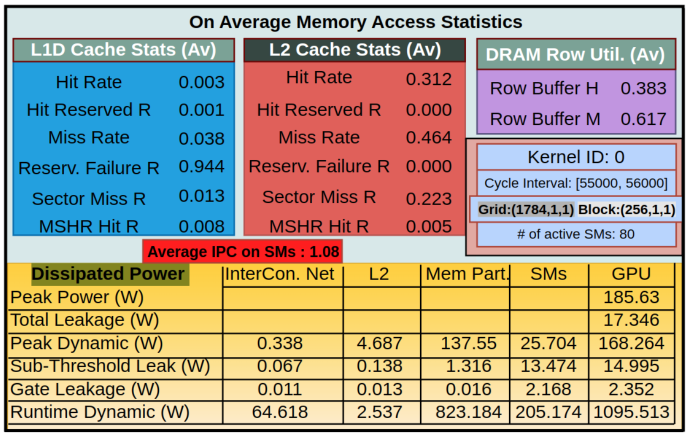
The comprehensive results of the dissipated power on GPU yield the analytical observation that gains significance, especially on embedded systems. Therefore, we develop GPPRMon to systematically collect the power distribution on the sub-units of SMs, memory partitions, and the interconnection network, in addition to performance metrics. Moreover, SMs are GPUs' most impactful hardware units in terms of runtime power dissipation with their dense compute units. Thus, GPPRMon further classifies SM's power distribution through execution units, including the register file and the beginning of the instruction pipeline, functional units, and load/store units involving the L1D cache.

We implement the collection of power metrics utility on top of the AccelWattch [14], built upon McPAT [21], and the accuracy and theoretical limitations of AccelWattch yet reside. However, GPPRMon's power metric collection feature is still a reliable and practical tool since we extend various configurability options of AccelWattch to GPPRMon, such as DVFS. GPPRMon assures the following runtime power measurements for each component apart from idle SM: *Peak Dynamic*(W), the maximum momentary power within the interval, *Sub-threshold Leakage (W)* and *Gate Leakage(W)*, the leaked power (due to current leakage) from the junctions of MOSFETs, and *Runtime Dynamic (W)*, the total consumed power. *Runtime Dynamic* power actually stands for the instantaneous power at the cycle granularity. However, since GPPRMon samples metric counters for each observation interval, this metric accumulates the power of each corresponding observation interval. Moreover, GPPRMon supports collecting power metrics either cumulatively or distinctly for each sampling interval, starting from a kernel's execution. For instance, power dissipation results in Figure 4 show the *cumulatively* collected results for the interval between [55000, 56000], which means aggregating power results for each sampling interval. The cumulative power metric collection option eases determining average power dissipation for different execution intervals. While V100's TDP is 300W [35], aggregated *Runtime Dynamic* power for 1000 cycles is 1095.5W, which corresponds to an average of 109.5W per sampling interval with the cycling frequency of 100 on the simulator. Users can configure the power metric collection by first enabling power profiling and determining other configurations as detailed in the tool's source code.

3.2. Visualization

By processing the collected metrics (Part $\boxed{1}$ in Figure 3), GPPRMon depicts performance and power dissipation with three perspectives at runtime, as represented in Part $\boxed{2}$ in Figure 3, and enables pointing out detailed interaction of the program with the underlying hardware.

i **General View**, Part $\boxed{\text{2-a}}$, presents the overall IPC of GPU, the average memory access statistics, and dissipated power among the major components with application- and architecture-specific information;
ii **Spatial View**, Part $\boxed{\text{2-b}}$, presents the detailed access statistics of the GPU memory hierarchy and dissipated overall power among the memory partitions by enabling the monitoring of the entire GPU memory space;
iii **Temporal View**, Part $\boxed{\text{2-c}}$, demonstrates instruction execution statistics with activation intervals at warp-level for user-specified thread blocks, L1D cache access characteristics, and power distribution among the sub-components of SMs by activating the execution monitoring feature.


On Average Memory Access Statistics

L1D Cache Stats (Av)	
Hit Rate	0.003
Hit Reserved R	0.001
Miss Rate	0.038
Reserv. Failure R	0.944
Sector Miss R	0.013
MSHR Hit R	0.008

L2 Cache Stats (Av)	
Hit Rate	0.312
Hit Reserved R	0.000
Miss Rate	0.464
Reserv. Failure R	0.000
Sector Miss R	0.223
MSHR Hit R	0.005

DRAM Row Util. (Av)	
Row Buffer H	0.383
Row Buffer M	0.617

Kernel ID: 0
Cycle Interval: [55000, 56000]
Grid:(1784,1,1) Block:(256,1,1)
# of active SMs: 80

Average IPC on SMs : 1.08

Dissipated Power	InterCon. Net	L2	Mem Part.	SMs	GPU
Peak Power (W)					185.63
Total Leakage (W)					17.346
Peak Dynamic (W)	0.338	4.687	137.55	25.704	168.264
Sub-Threshold Leak (W)	0.067	0.138	1.316	13.474	14.995
Gate Leakage (W)	0.011	0.013	0.016	2.168	2.352
Runtime Dynamic (W)	64.618	2.537	823.184	205.174	1095.513

Fig. 4. *General View* with average performance and power consumption metrics collected at runtime

GPPRMon includes three configuration options for different visualization perspectives and an interval sampling cycle to divide runtime execution into portions. Since GPPRMon's *Temporal View* may require scanning of many statistics for all thread blocks, especially in large architectures, GPPRMon provides a *Temporal View* option among the thread blocks determined with IDs. Depending on the configuration, GPPRMon starts tracking collected metrics for each kernel and systematically saves images in PNG format per execution interval.

Figure 4, an example of our *General View*, presents the overall measurement of the first kernel for the *SPMV* from *Gardenia* benchmark [48] executed on V100 GPU [23]. It displays average memory access statistics among the active L1D caches, L2 caches, and

DRAM banks; average IPC value among the active SMs; dissipated power on major sub-GPU components within the [55000, 60000] cycle interval. The view includes grid (i.e., 1764 thread blocks) and block dimensions (i.e., 256 threads per block) with the number of actively used SMs so that the users can relate active SMs and workloads at runtime. To illustrate, Figure 4 shows that the kernel executes with the IPC rate of 1.08 and utilizes the memory hierarchy inefficiently due to high miss and reservation failure rates on V100 in this interval, where the Volta SM architecture supports concurrent execution of 2048 threads per SM. Considering that V100 SMs contain 16 SP/INT/DP ALUs, SFUs, and Tensor Cores, we can notice the low performance since the ideal IPC should be much more than 1.08 with 640 thread blocks (8 thread blocks per SM) in the given interval. Long-latency memory operations may slow instruction completion and result in low IPC. Moreover, memory partitions consume 75% of total power dissipation, which validates that SMs mostly stay idle for the execution interval given in Figure 4.

Our *General View* supports two additional visuals that show the time spanning of memory access statistics and the relationship between IPC and power metrics for longer runtime intervals as in Figure 11 and Figure 14 (the examples given as part of our case studies), respectively.

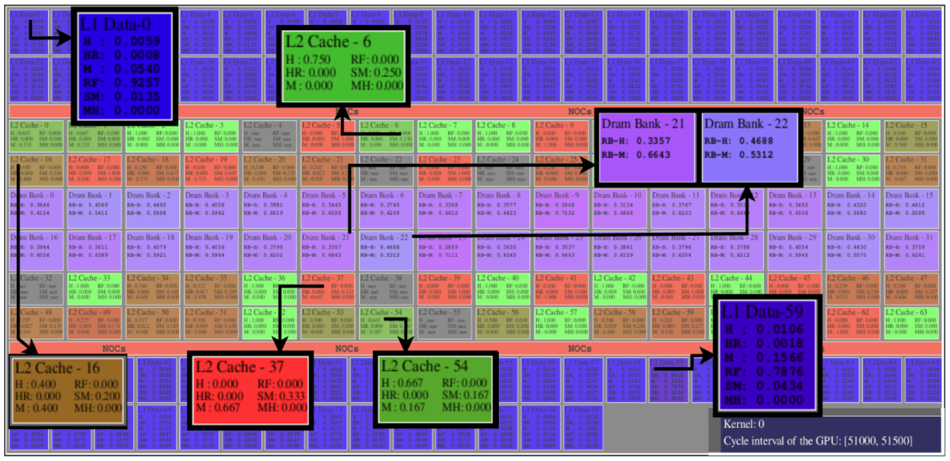
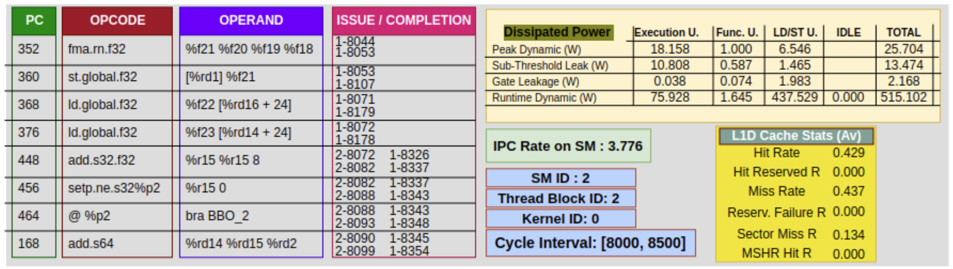


Fig. 5. *Spatial View* displaying memory access statistic at runtime

Figure 5, an example of our *Spatial View*, shows the memory access statistics across the GPU memory hierarchy on L1D caches, L2 caches, and DRAM. On caches, the green emphasizes hit and hit reserved accesses, the red indicates miss and sector miss, and the blue states the reservation failures through miss queues or MSHR. Similarly, DRAM bank pixels are colored with a mixture of red and blue to specify the row buffer misses and hits, respectively. GPPRMon supports memory hierarchy visualization for all official GPU configurations of the simulator, even if some disable the L1D cache for load/store operations. *Spatial View* includes detailed data about access statistics, resource quantities, architecture types, and dissipated power on memory partitions. To detail the view, we zoom in on some memory units in Figure 5, which presents statistics in the cycles of [51000, 51500] for the kernel execution. Different L1D caches behave similarly in that interval, such that almost all L1D caches turn blue due to reservation failure concentration. To illustrate, the reservation failure rates are 0.92 and 0.78 on *L1D-0* and *L1D-1*, respectively. On the contrary, statistics on L2 caches imply heterogeneous data accesses. While *L2 Cache-6* and *L2 Cache-54* bring hit rates of 0.75 and 0.67, respectively, *L2 Cache-37* causes 0.67 miss and 0.33 sector miss rates, and L2 cache is intermediate in utilizing data locality among the others. Gray color among the units regards no access occurring on the *Spatial View*.



PC	OPCODE	OPERAND	ISSUE / COMPLETION
352	fma.rn.f32	%f21 %f20 %f19 %f18	1-8044 1-8053
360	st.global.f32	[%rd1] %f21	1-8053 1-8107
368	ld.global.f32	%f22 [%rd16 + 24]	1-8071 1-8179
376	ld.global.f32	%f23 [%rd14 + 24]	1-8072 1-8178
448	add.s32.f32	%r15 %r15 8	2-8072 1-8326 2-8082 1-8337
456	setp.ne.s32%p2	%r15 0	2-8082 1-8337 2-8088 1-8343
464	@ %p2	bra BBO_2	2-8088 1-8343 2-8093 1-8348
168	add.s64	%rd14 %rd15 %rd2	2-8090 1-8345 2-8099 1-8354

Dissipated Power	Execution U.	Func. U.	LD/ST U.	IDLE	TOTAL
Peak Dynamic (W)	18.158	1.000	6.546		25.704
Sub-Threshold Leak (W)	10.808	0.587	1.465		13.474
Gate Leakage (W)	0.038	0.074	1.983		2.168
Runtime Dynamic (W)	75.928	1.645	437.529	0.000	515.102

IPC Rate on SM : 3.776

SM ID : 2

Thread Block ID: 2

Kernel ID: 0

Cycle Interval: [8000, 8500]

L1D Cache Stats (Av)	
Hit Rate	0.429
Hit Reserved R	0.000
Miss Rate	0.437
Reserv. Failure R	0.000
Sector Miss R	0.134
MSHR Hit R	0.000

Fig. 6. *Temporal View* monitoring instruction executions together with the performance and power consumption of the corresponding SM

Figure 6, an example of our *Temporal View*, displays a thread block's execution statistics at warp-level, L1D cache statistics, and dissipated power of core components configurable execution intervals. It presents each warp's PTX instruction sequence, with opcodes, operands (source/destination registers and immediate values if they exist), and the program counter (PC). The *Issue/Completion* column indicates the execution start and writeback times of warp instruction segments within any thread block. For instance, Figure 6 reveals the execution monitoring of the *Thread Block 2* on *SM 2* in the cycle range of [8000, 8500]. The instruction dispatcher unit issues two SP global loads with PC=368 and PC=376 at *Cycle 8071* and *Cycle 8072*, and they are completed at *Cycle 8179* and *Cycle 8178*, respectively. *Temporal View* allows tracking execution duration per instruction in this manner. Since L1D cache hits result in low latency, these load instructions lasting above 100 cycles are among the misses or sector misses of L1D cache statistics. In addition, the view enables us to relate IPC, instruction statistics, and power metrics. The fact that the rate of memory instructions is 0.37 and inefficient use of the L1D cache within the given interval significantly degrades the IPC on SM2. Furthermore, the load/store unit dissipates nearly 92% of SMs total power in the corresponding interval because of the pressure on the L1D cache. As a result, one can analyze the data locality in a multi-perspective by utilizing access statistics of caches and row buffers on *Spatial View*, tracing the issue/completion times of memory operations on Temporal View at runtime.

GPPRMon Overheads. GPPRMon execution performance mainly depends on ***i.*** *the metric collection sampling frequency*, ***ii.*** *visualizer intervals*, and ***iii.*** view types. Firstly, each sampling operation during simulation conducts multiple I/O operations to the metric collection files, such as performance, memory, and power trace files, which are significantly slow compared to the simulation execution. Hence, widening the simulation runtime sampling interval directly reduces the number of I/O operations for exporting results to output files, and the simulation duration decreases accordingly. For instance, simulating the *SPMV* benchmark [48] takes 98 minutes for the *Higgs Twitter Mention* data by recording both power and performance metrics per 5000 simulation cycles, while the baseline simulation (i.e., not collecting runtime performance and power metrics) completes the execution at 88 minutes in our local infrastructure. Furthermore, the visualization interval mainly determines the spanning range of the collected metrics. That is, lower visualization intervals search for fewer sampled metrics, reducing generating views to demonstrate runtime performance and power observations on the GPU. Since each view type requires different metrics, composing each figure takes various amounts of time. For example, generating *General View* in Figure 4 necessitates spanning all statistics on L1D and L2 caches, row buffers, SMs, and power dissipation for a given internal while *Spatial View* only displays memory access statistics, and is much easier to compose. Additionally, tracking the instruction monitoring of limited thread blocks through *Temporal View* is comparably easy, whereas increasing the number of thread blocks for instruction monitoring raises the spanning overhead and takes longer to generate those views.

4. Usage Scenarios

We evaluate a set of CUDA programs from two benchmark suites and demonstrate the case studies that can use our framework. Specifically, we utilize graph workloads from the Gardenia benchmark suite [48] and embedded applications from the GPU4S embedded benchmark suite [37]. Since graph workloads target large-scale systems, we configure the GPPRMon to simulate Volta architecture-based V100, commonly used in HPC systems and GPU architecture research. Table 1 presents salient characteristics of the V100 device. On the other hand, for embedded applications, we configure our tool to simulate the GPU device on Jetson AGX Xavier, which provides a System-on-Module with a Volta-based GPU and contains an 8GB/16GB unified memory with a high-bandwidth interface to the GPU, and a 512KB shared L2 cache exists in the memory hierarchy [25]. Its GPU includes 512 cores corresponding to 8 SMs involving a 128KB on-chip cache per SM.

Table 1. V100 GPU configuration specifications based on Volta architecture

SM (80) Specifications	Registers, Register Banks	65536, 16
	SP-U, SF-U, DP-U, INT-U, TC-U, LD/ST-U (WB-PipeDepth)	4,4,4,4,4,1(8)
	Warp Scheduler	4
	on-chip L1I Cache, NoF banks, access latency, cache line	128KB (64 sets, 16-way), 1, 20 cycles, 128B
Memory Parititon (32) Specifications	L2 Cache, NoF banks, access latency, cache line	96KB (32 sets and 24-way), 2, 160 cycles, 128B
	DRAM, NoF banks, access latency (after L2)	1GB, 16 banks, 100 cycles, 128B
	DRAM scheduler	First-ready, first-come first-service

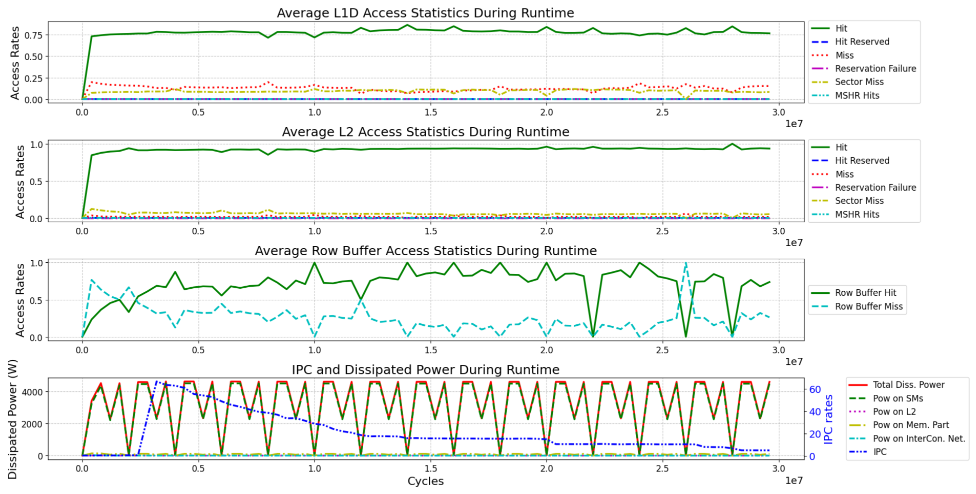


Fig. 7. Average memory and power consumption statistics for *BC-Kernel 1*

4.1. Determining Memory-Bound Kernels and Spatial Characteristics

We evaluate the *Betweenness Centrality (BC)* program (from Gardenia suite [48]) with multiple kernel executions. By analyzing the kernels with the largest cycles, we classify them as kernels with no memory pressure and kernels with memory bottleneck. Additionally, we review SM distribution and utilization of the target executions.

Figure 7 presents memory-related data and IPC values during the execution time intervals for the kernel function, *Kernel 1*, and demonstrates L1, L2, and row buffer statistics and the corresponding IPC/power behavior. While our spatial and temporal views present fine-grained values for each hardware component across the GPU device, those statistics (Figure 7 and Figure 9 afterward) demonstrate the dynamic average values for overall GPU SMs. While there are a few oscillations in the rates, the general behavior demonstrates steadily high hit rates. Moreover, we can see the peak IPC value during the intensive computations at the beginning of the kernel execution. Then, IPC values tend to decrease as the computations end. We can observe that the kernel has no memory pressure during the execution. After warming the caches at the very beginning of the execution, the kernel could perform its operations with high hit rates and compatible IPC rates.

After getting the memory behavior of our target kernel based on our average timing statistics, we can utilize our *Spatial View* to understand its SM utilization. Figure 8 presents the partial view that includes the L1 cache structures of each SM at time intervals [5000-10000], [15000-20000], and [245000-250000], respectively. The application launches the kernel function with 64 thread blocks and 256 threads per block, such that the thread blocks are scheduled to 64 SMs (out of 80 in the V100 device). Therefore, only the corresponding L1 caches exhibit non-zero values. Since all thread blocks are active and the caches do not hold data at the beginning (5K-10K time interval), all 64 L1 cache structures employ low hit rates (red color in our representation, while inactive L1s are gray). At the next time interval, the execution starts warming the caches, and hit rates increase (green color in our representation). Eventually, as the thread blocks complete execution and they retire, L1 caches on the corresponding SMs do not collect statistics

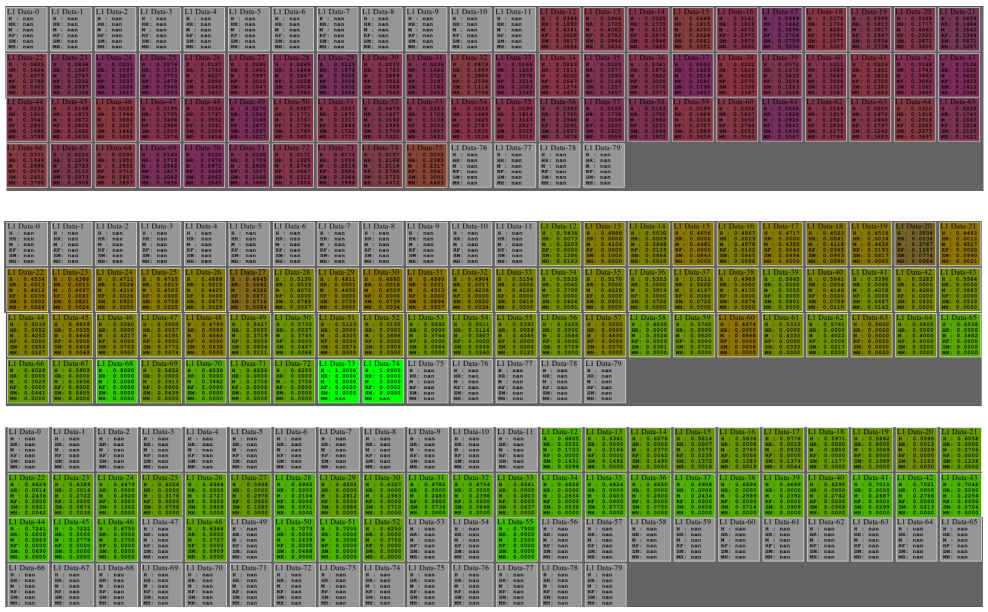


Fig. 8. L1 cache status for *BC-Kernel 1* for successive time intervals

(gray as the other L1 caches at the beginning). We can track all thread blocks' scheduling and their L1 cache utilization during the execution for the given time intervals. For the specific kernel execution, observing the high hit rates (green colors) in all L1 caches of the active SMs and corresponding IPC rates demonstrates efficient L1 utilization and low memory pressure.

Figure 9 demonstrates memory-related data and IPC values for another kernel, *Kernel 2*. Since *Kernel 2* has been launched by much more threads than available SMs, all SMs are active during the execution, and the threads compete for both L1 and L2 caches. The reservation failure and miss rates at the first part of the execution are significant for private L1 and shared L2 cache, in turn, the kernel exhibits low IPC values. Whenever L2 and especially L1 hit rates become more than 0.5 (around cycle 600K), IPC values also increase, and the kernel function performs most of the target computations (until cycle 800K). We can observe the corresponding cache status in our *Spatial View*. For this case, we include both L1 and L2 cache structures in our aggregated visualization in Figure 10. Different from *Kernel 1*, we can see that all L1 caches exhibit non-zero values from the beginning of the execution since the available threads utilize all SMs in the device. The figure demonstrates the warming of the caches through the execution cycles, where the kernel function has pressure on both L1 and L2 by highly utilizing those structures.

4.2. Performance Bottleneck Analysis and its Power Impacts for a Memory-Intensive Workload

We evaluate CUDA implementation of the *Page Ranking (PR)* algorithm given in Gardenia suite [48] to analyze a memory-bound GPU program and irregular memory access statistics through GPPRMon. *PR* assigns weights to graph nodes describing relative importance among nodes.

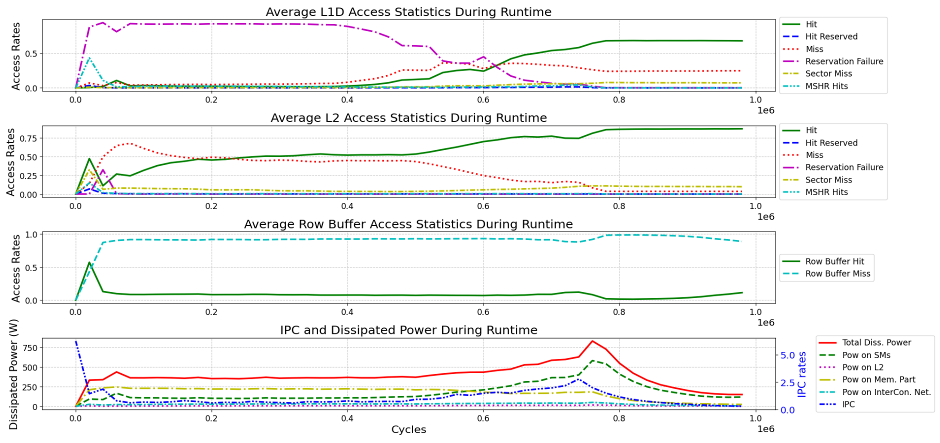


Fig. 9. Average memory and power consumption statistics for *BC-Kernel 2*

The *PR* execution iterates with the *Contribution Step (K0)*, *Pull Step (K1)*, and *Linear Normalization (K2)* kernels, and the total number of iterations varies depending on the data size. Table 2 presents the performance overview of the kernel statistics. At the beginning of the execution, *K0* causes a high miss rate on caches since all memory operations are completed before warming up. No row buffer locality information is provided across DRAM accesses, as the required data mostly fits into the L2 cache. In other words, L2 misses do not access the same row of DRAM banks within any memory partition. Since total elapsed cycles indicate that *K1* dominates the execution at 99.7% and directly affects the application performance and overall power consumption, we focus on that kernel execution.

Table 2. *Page Ranking (PR)* kernel performance statistics

Kernel	**GPU IPC**	**GPU Occupancy**	**L1D**		**L2**		**DRAM**		**Total Cycle**
			Miss Rate	Res. Fail Rate	Miss Rate	Res. Fail Rate	Row B. Loc. (L+S)	Row B. Loc. (Load)	
Page Ranking - Contrib K0	715.59	82.76%	1.000	0.819	0.333	0.0	NA	NA	8670
Page Ranking - PullStep K1	3.007	5.55%	0.584	0.400	0.156	0.011	0.658	0.667	8677889
Page Ranking - LinNorm K2	1297.68	77.108%	0.501	0.285	0.457	0.001	0.724	0.732	11718

V100 includes 870GB/s bandwidth migrating 217.5 Giga SP float to the SMs and 14.8-SP/7.4-DP TFLOPS peak computational power [23]. With these specifications, we can state that the time consumed for one load complies with the execution of 68 SP float operations on SMs, ideally (i.e., without L1D and L2 caches). On the other hand, *K1* has a memory instruction intensity of around 0.2 in its PTX code, which validates that *K1* bounds the performance due to intrinsic memory workload. Not only is *K1* memory-bound, but also inefficient memory usage severely impacts performance. To see performance-

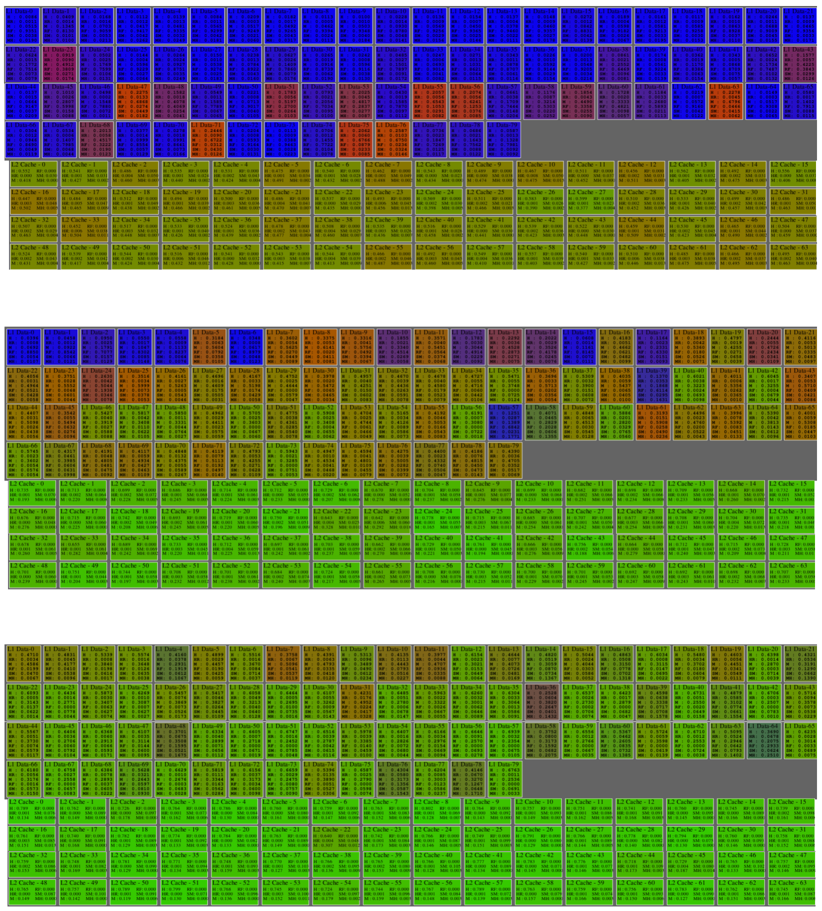


Fig. 10. L1 and L2 cache status for *BC-Kernel 2* from cycles 400K, 600K, 700K

critical points in the execution, we monitor runtime performance-degrading factors with low-frequency execution snapshots provided as part of our tool.

While the overall IPC is around 0.3 for *K1*, *General View* results obtained for every 500 cycles during the execution indicate that IPC oscillates in the range of [0.1, 9.54], with the highest peak of 53.44. Figure 11, which is part of our *General Overview*, shows average access statistics on memory units in [5000, 100000] cycles. After caches warming up (around cycles 10000), while the average miss rate on L1D caches oscillates in [0.14, 0.51], sector misses, which the simulator does not provide separately, vary in [0.05, 0.31] with the metrics collected in every 20 cycles. We can understand the data pollution on the L1D caches, which prevents the execution from exploiting cache locality. For instance, *K1* does not utilize spatial locality on the L1D cache since the MSHR hits oscillate slightly in [0.03, 0.08] during the execution. Hence, the overall hit rate on L2 caches is quite high when we use the web-Stanford dataset [19], which occupies memory space

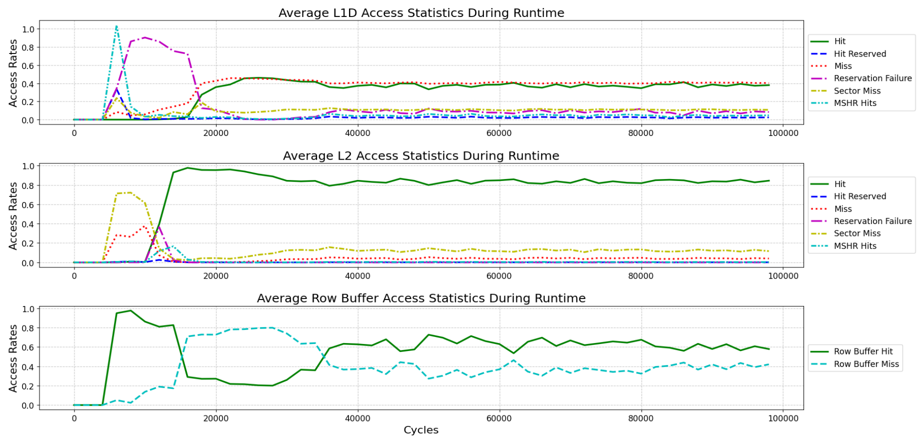
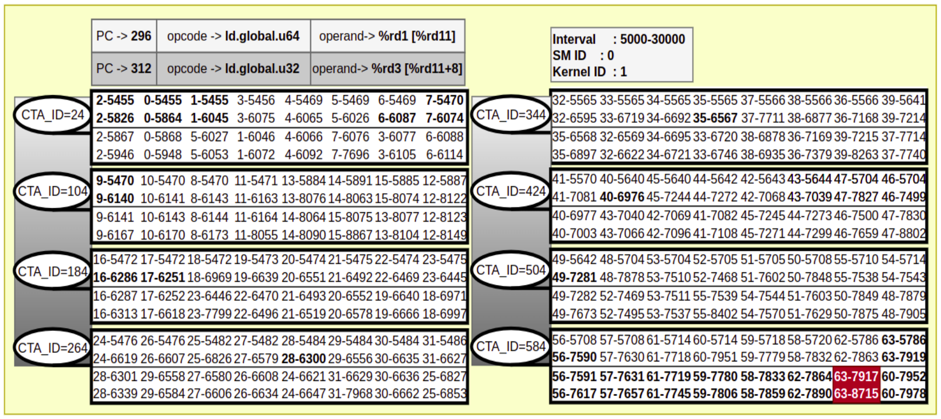


Fig. 11. Average memory access statistics in the cycle range of [5000, 100000] for *PR-K1*

five times larger than the L2 cache size. While the performance metrics in Table 2 hide the L2 statistics as it counts misses before warming up at kernel launch, the actual L2 hit rate oscillates in [0.82, 0.95] at runtime with sampling per 500 cycles. Additionally, the row buffer hits and misses vary in [0.2, 0.85] in an unstable manner, which verifies data sparsity throughout the execution.



PC -> **296**	opcode -> **ld.global.u64**	operand-> **%rd1 [%rd11]**
PC -> **312**	opcode -> **ld.global.u32**	operand-> **%rd3 [%rd11+8]**

Interval : 5000-30000
SM ID : 0
Kernel ID : 1

CTA_ID=24	**2-5455**	**0-5455**	**1-5455**	3-5456	4-5469	5-5469	6-5469	**7-5470**
	2-5826	**0-5864**	**1-6045**	3-6075	4-6065	5-6026	**6-6087**	**7-6074**
	2-5867	0-5868	5-6027	1-6046	4-6066	7-6076	3-6077	6-6088
	2-5946	0-5948	5-6053	1-6072	4-6092	7-7696	3-6105	6-6114
CTA_ID=104	**9-5470**	10-5470	8-5470	11-5471	13-5884	14-5891	15-5885	12-5887
	9-6140	10-6141	8-6143	11-6163	13-8076	14-8063	15-8074	12-8122
	9-6141	10-6143	8-6144	11-6164	14-8064	15-8075	13-8077	12-8123
	9-6167	10-6170	8-6173	11-8055	14-8090	15-8867	13-8104	12-8149
CTA_ID=184	16-5472	17-5472	18-5472	19-5473	20-5474	21-5475	22-5474	23-5475
	16-6286	**17-6251**	18-6969	19-6639	20-6551	21-6492	22-6469	23-6445
	16-6287	17-6252	23-6446	22-6470	21-6493	20-6552	19-6640	18-6971
	16-6313	17-6618	23-7799	22-6496	21-6519	20-6578	19-6666	18-6997
CTA_ID=264	24-5476	26-5476	25-5482	27-5482	28-5484	29-5484	30-5484	31-5486
	24-6619	26-6607	25-6826	27-6579	**28-6300**	29-6556	30-6635	31-6627
	28-6301	29-6558	27-6580	26-6608	24-6621	31-6629	30-6636	25-6827
	28-6339	29-6584	27-6606	26-6634	24-6647	31-7968	30-6662	25-6853

CTA_ID=344	32-5565	33-5565	34-5565	35-5565	37-5566	38-5566	36-5566	39-5641
	32-6595	33-6719	34-6692	**35-6567**	37-7711	38-6877	36-7168	39-7214
	35-6568	32-6569	34-6695	33-6720	38-6878	36-7169	39-7215	37-7714
	35-6897	32-6622	34-6721	33-6746	38-6935	36-7379	39-8263	37-7740
CTA_ID=424	41-5570	40-5640	45-5640	44-5642	42-5643	**43-5644**	**47-5704**	**46-5704**
	41-7081	**40-6976**	45-7244	44-7272	42-7068	**43-7039**	**47-7827**	**46-7499**
	40-6977	43-7040	42-7069	41-7082	45-7245	44-7273	46-7500	47-7830
	40-7003	43-7066	42-7096	41-7108	45-7271	44-7299	46-7659	47-8802
CTA_ID=504	49-5642	48-5704	53-5704	52-5705	51-5705	50-5708	55-5710	54-5714
	49-7281	48-7878	53-7510	52-7468	51-7602	50-7848	55-7538	54-7543
	49-7282	52-7469	53-7511	55-7539	54-7544	51-7603	50-7849	48-7879
	49-7673	52-7495	53-7537	55-8402	54-7570	51-7629	50-7875	48-7905
CTA_ID=584	56-5708	57-5708	61-5714	60-5714	59-5718	58-5720	62-5786	**63-5786**
	56-7590	57-7630	61-7718	60-7951	59-7779	58-7832	62-7863	**63-7919**
	56-7591	**57-7631**	**61-7719**	**59-7780**	**58-7833**	**62-7864**	**63-7917**	**60-7952**
	56-7617	**57-7657**	**61-7745**	**59-7806**	**58-7859**	**62-7890**	**63-8715**	**60-7978**

Fig. 12. Instruction monitoring for the load instructions on *SM 0* in the cycle range of [5000, 30000] for *PR-K1*

Figure 12 presents the instruction issue/completion cycles of 8 *K1* thread blocks running on *SM 0*. We merge multiple snapshots of our *Temporal View* that belong to thread blocks in *SM 0* to evaluate the performance of all load instructions together. The first and second lines point to the load instructions of which the program counter (PC) equals 296 (loads DP) and 312 (loads SP), respectively. Figure 13, a snapshot of our *Spatial View*, demonstrates the memory access statistics of representative components within the same interval. After the kernel launch, each thread collects thread-specific information from

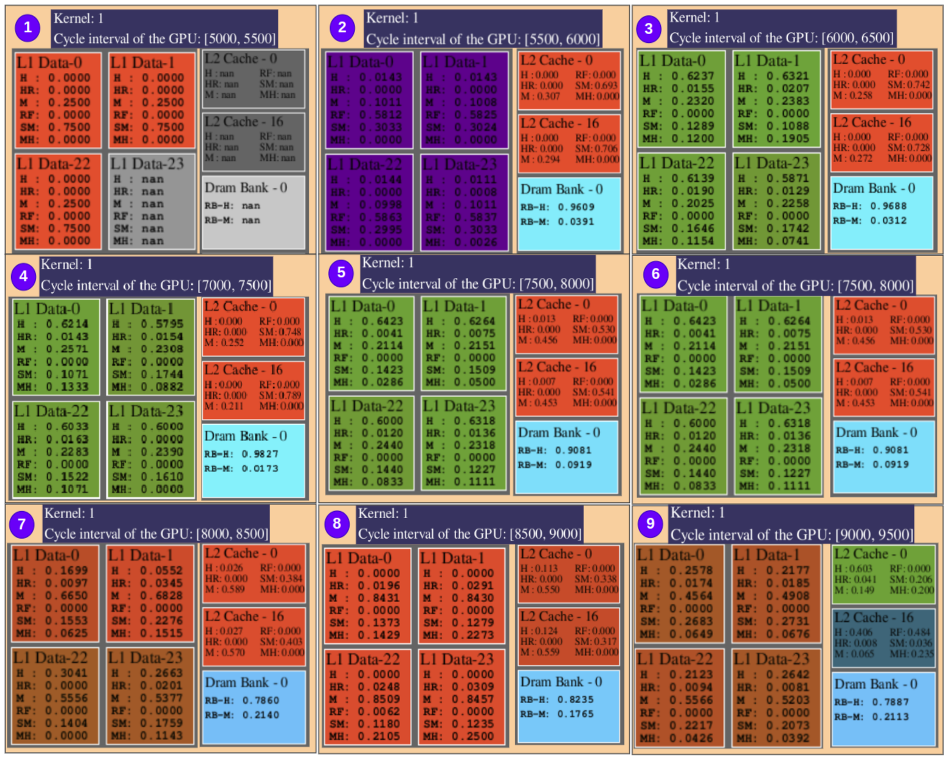

Kernel: 1
Cycle interval of the GPU: [5000, 5500]
Cycle interval of the GPU: [5500, 6000]
Cycle interval of the GPU: [6000, 6500]
Cycle interval of the GPU: [7000, 7500]
Cycle interval of the GPU: [7500, 8000]
Cycle interval of the GPU: [8000, 8500]
Cycle interval of the GPU: [8500, 9000]
Cycle interval of the GPU: [9000, 9500]
L1 Data-0
L1 Data-1
L1 Data-22
L1 Data-23
L2 Cache - 0
L2 Cache - 16
Dram Bank - 0

Fig. 13. Memory performance overview in the cycle range of [5000, 9500] for *PR-K1*

parameter memory, which takes 250-450 cycles to process target data addressed with the thread's private registers. The warp schedulers dispatch the load instructions (i.e., at PC=296 *ld.global.u64*), and all eight warps at *Thread Block 24* start executing the instruction after *Cycle 5455*. Furthermore, Figure 12 reveals that SM dispatches load instructions from the remaining thread blocks in the interval of [5470, 5786] after issuing the load instructions of *Thread Block 24*. Figure 13 reveals that no access occurs on some of the L1D caches, while none of the L2 caches and DRAM banks are accessed during the preparation time in [5000, 5500] in Part $\boxed{1}$. No data brought to the L1D cache of *SM 0* by the warps of *Thread Block 24* after *Cycle 6087* (*Warp 6*) enables the early completion of the instruction pointed with PC=296, belonging to the thread blocks 104, 184, 264, 344, 424, 504, and 584. We highlight this observation with the bold fonts representing the earliest completion times within each thread block in the second row of the first instructions. Additionally, a high reservation failure and no MSHR hit rates on the L1D cache of *SM 0* in Part $\boxed{2}$ confirms that the locality utilization between thread blocks is quite low for the first load. If the L1D cache locality was utilized, we should have observed larger MSHR hit rates and completion of the same instructions just after *Cycle 6087*. Parts $\boxed{2,3,4,5}$ reveal that the reservation failed requests pointed by PC=296 cause a miss on L2 caches without MSHR merging. Thus, memory requests of the same instruction from different SMs cannot benefit from L2 locality and cause more traffic in the memory hierarchy. Additionally, Parts $\boxed{3,4,5,6}$ reveal that the L1 access status mostly turns to the hit after *Cycle 6000*. Unlike the load instructions at PC=296, ones at PC=312 (*ld.global.u32*) usually hit on L1. The second line for each thread block shown in Figure 12 indicates that the completion takes much fewer cycles for the loads at PC=312. To illustrate, while *Thread Block 504* completes the first load instructions within 2133 cycles, except *Warp 63*, it takes 26 cycles for the second instructions, whose requests result in a miss on both the L1D and L2 caches. While the loads at PC=296 complete the execution in the range of [350, 2250] cycles, the loads at PC=312 take less than 50 cycles for most of the warps due to the increasing hits on the L1D cache. However, the loads at PC=296 delay the issue of the second load instructions due to long latency. The remaining thread blocks (from *Thread Block 641*) are assigned to SMs after *Cycle 55000*. With the observation that a thread block occupies 50000 cycles on an SM (for the web-Stanford graph, which can easily fit into DRAM), the schedule of any waiting thread block delays around 2000 cycles. Such delays affect the performance of a thread block by 4% in addition to affecting the memory traffic negatively and degrading the overall performance of *K1*. In this manner, an approach such as adaptive thread block scheduling by throttling the load/store unit issue amount depending on the access statistics of caches can reduce the side effects and increase the overall performance.

Table 3. Dissipated average power in Watts in the cycle range of [5000,10000] for *PR-K1*

	Streaming Multiprocessors					**Memory Partitions**							
Cycles	**Exec. Units**	**Func. Units**	**LD/ST Unit**	**Idle**	**Total**	**MC FEE**	**PHY**	**MC TE**	**Dram**	**L2**	**Total**	**NoC**	**TOTAL**
5k, 5.5k	2637.5	54.3	35.6	23.7	2751.3	3.7	8.2	4.6	0	0	16.5	0.7	2768.5
5.5k, 6k	597	6.3	860	0	1463.7	177.2	17.8	9.4	557.2	3.4	764.9	26.9	2501.6
6k, 6.5k	614.4	12.4	399.9	0	1026.7	56.1	31.3	16.1	1346.4	3.0	1452.9	92.6	2577.3
6.5k, 7k	708.6	14.4	464.3	0	1187.3	65.5	31.4	16.2	1354.3	3.1	1470.5	94.2	2755.1
7k, 7.5k	686.8	13.9	463.9	0	1164.6	65.6	31.8	16.2	1354.9	3.1	1471.2	94.4	2733.2
7.5k, 8k	795.8	16.3	487.4	0	1299.4	69.9	29.7	15.3	1264.3	3.7	1383.1	96.7	2782.9
8k, 8.5k	543	10.2	335	0	888.2	60.4	30.5	15.7	1341.4	4.4	1452.0	124.7	2469.3
8.5k, 9k	354.5	5.6	249.3	0	609.3	52	31.1	16.1	1362.6	19	1480.7	148	2257.2
9k, 9.5k	474.9	5.3	455.4	0	935.7	80.2	27.8	14.4	1096.9	66.1	1284.4	216.8	2503
9.5k, 10k	446.8	4.8	475.5	0	927	78.2	23.8	12.4	843.2	41	968.7	153.8	2090.4

Table 3 presents power measurements for *K1* execution throughout the hardware structures. After kernel launch, (i.e., *Cycle 5000*), the threads load thread-identifier data from the parameter memory, causing higher power consumption on SMs. The power values of the memory partitions in the following cycles get higher than the SMs. Additionally, the power consumption by the LD/ST unit is high due to the intense memory operations and pressure on the L1D cache, and other units apart from the register file portion of the execution unit get lower after *Cycle 5500*. We can see that DRAM consumes the most power in the memory partitions with intense usage of high-bandwidth [24].

4.3. Performance-Power Analysis of an Embedded Application

Embedded applications targeting artificial intelligence, computer vision, and advanced graphics require high computational power. Jetson AGX Xavier provides a System-on-

Module that meets these demands with a Volta-based GPU. We execute CUDA implementation of the *Fast Fourier Transform (FFT)* from the GPU4S suite [37].

Table 4 presents the overall performance metrics for executing the first three kernels. The thread blocks consist of 256 threads, and a maximum of 8 thread blocks can run in parallel on the SMs since the register file limits the number of simultaneous thread block execution. Since the kernels other than the *K0* utilize the hardware similarly, they result in similar performance as shown for *K1* and *K2*. Thus, we explain the relationship between performance and power consumption for *K0*, which employs diverse characteristics.

Table 4. Fast Fourier kernel performance statistics

Kernel	GPU IPC	GPU Occ.	L1D		L2		DRAM		Total Cycle
			Miss Rate	Res. Fail Rate	Miss Rate	Res. Fail Rate	Row Buffer Loc. (L+S)	Row Buffer Loc. (Load)	
FFT - K0	31.76	62.52 %	0.67	0.92	0.65	0	0.21	0.89	990276
FFT - K1	100.22	80.25 %	0.1	0,75	0.20	0	0.33	0.92	235390
FFT - K2	49.20	85.28 %	0.1	0.837	0.21	0	0.41	0.92	239755

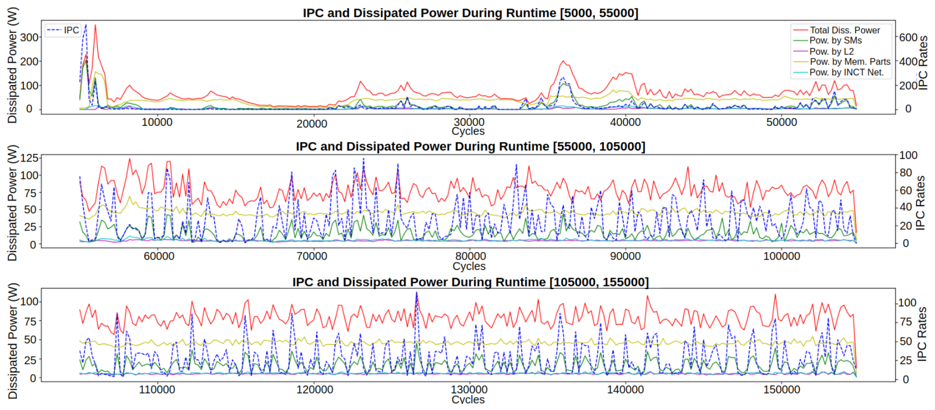
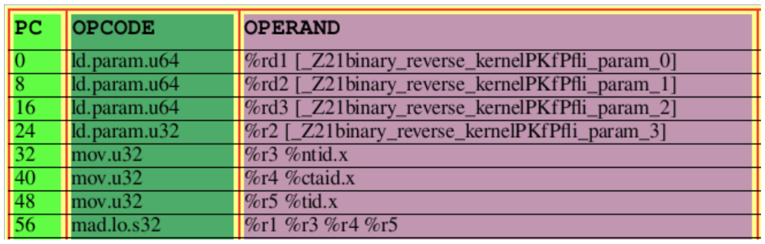


Fig. 14. IPC with dissipated power metrics in the cycle range of [5000, 155000] for *FFT-K0*

Figure 14 displays the IPC and power metrics measurements for *K0* at different execution cycle intervals. To avoid losing observation details related to IPC and power metrics of the kernel, we report the relationship in three execution intervals separately. At cycles [5000, 6500], the power per cycle and IPC values increase significantly since each SM registers thread-identifier information in their private registers, causing a high activation on register files and functional units. Figure 15, (as part of our *General View*), shows four loads, three data movements, and one multiply-add instruction executed by 16384 threads concurrently (with 64 thread blocks where each contains 256 threads) in that interval. Throughout the *K0* execution, we have high L1D and L2 miss rates. Figure 14 reveals the effect of those accesses on power dissipation (yellow line). The memory partitions dissipate half of the total power, around 50W, due to intensive activation at runtime. IPC and power dissipation increase instantly with SM activation points. By tracking the points, where IPC increases between [55000, 105000] and [105000, 155000] cycles, we can see parallel increments in power metrics dissipated by SMs and GPU. While the overall IPC value for *K0* equals 31.76 on SMs, runtime IPC varies in the range of [5, 75]. The concurrent load and store instructions cause small latencies and slight computational loads on the SMs.



PC	OPCODE	OPERAND
0	ld.param.u64	%rd1 [_Z21binary_reverse_kernelPKfPfli_param_0]
8	ld.param.u64	%rd2 [_Z21binary_reverse_kernelPKfPfli_param_1]
16	ld.param.u64	%rd3 [_Z21binary_reverse_kernelPKfPfli_param_2]
24	ld.param.u32	%r2 [_Z21binary_reverse_kernelPKfPfli_param_3]
32	mov.u32	%r3 %ntid.x
40	mov.u32	%r4 %ctaid.x
48	mov.u32	%r5 %tid.x
56	mad.lo.s32	%r1 %r3 %r4 %r5

Fig. 15. Instructions preparing threads for execution with thread-specific data in the cycle range of [5000, 6500] for *FFT-K0*

4.4. Analyzing the Impact of the Input Size on Resource-Critical Embedded System

Since embedded devices employ relatively smaller computational and memory resources, the target programs' resource utilization becomes more important and influential on the execution performance. Besides program characteristics, the input size significantly affects the pressure on the memory structures and the obtained performance. To understand the impacts of the input size on memory behavior and execution performance, we utilize the memory statistics view of our tool. We execute *softmax* program from GPU4S suite [37] for two matrix sizes (1024 and 4096).

Figure 16 presents the memory statistics of the first 500K cycles. The execution employs high L1 reservation failure rates and L2 miss rates for the first 300K cycles for both executions. However, the small input (1024) can fit within the L2 and L1 caches around *Cycle 300K* and *Cycle 350K*, respectively. On the other hand, the large input (4096) execution still employs low hit L1 and L2 rates.

Figure 17 demonstrates the corresponding IPC behavior, which is compatible with cache rates. While the IPC for the large input case oscillates at small values (0.2-0.8), the IPC values for the small input case are steadily low but increase after L2 and L1 cache breakpoints. With this analysis, we can decide on the cache utilization of the target input dimension for the program.

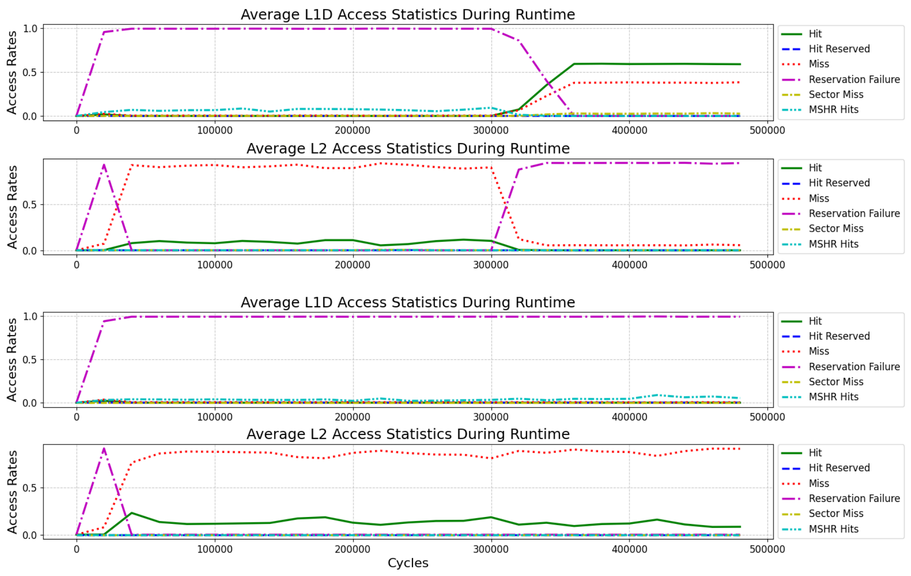


Fig. 16. Average memory access statistics at first 500K cycles for *softmax* kernel (matrix sizes with 1024 and 4096)

5. Related Work

In this section, we review the existing GPU execution analysis and visualization tools in the literature. For each work, we emphasize GPPRMon's fine-grain evaluation and visualization support for performance and power analysis by specifying the differences from the existing approach.

AerialVision [1] visualizes runtime warp divergence, dynamic IPC, global memory access statistics, active thread count, and a mapping window between source code and exposed pipeline latency metrics of kernel execution. While GPPRMon enables developers to dig into details of runtime GPU execution within specific microarchitectural units, AerialVision profiles the run time metrics on a much longer execution scale and visualizes overall GPU performance without per-component performance analysis. GPPRMon is similar to AerialVision in terms of providing runtime performance metrics, but GPPRMon can uncover more detailed behavioral insight inside kernel execution with more detailed runtime observation opportunities. Furthermore, AerialVision does not support displaying dissipated power.

Nsight Compute Tool [27] runs an application on an NVIDIA GPU device and collects average hardware usage statistics for the main components on a kernel basis. Nsight System Tool [28] is the other tool that is mostly preferred to analyze end-to-end kernel execution performance, including CPU-GPU communications. Profiling through these tools eases interpreting the overall kernel performance and hardware utilization with well-designed GUIs. In addition to performance analysis tools, GPU users can track instant power dissipation of deployed GPU through *System Management Interface* (SMI) library (i.e., nvidia-smi) [31]. Different from all NVIDIA tools and library frameworks, GPPRMon can track and visualize execution and hardware utilization statistics at run-

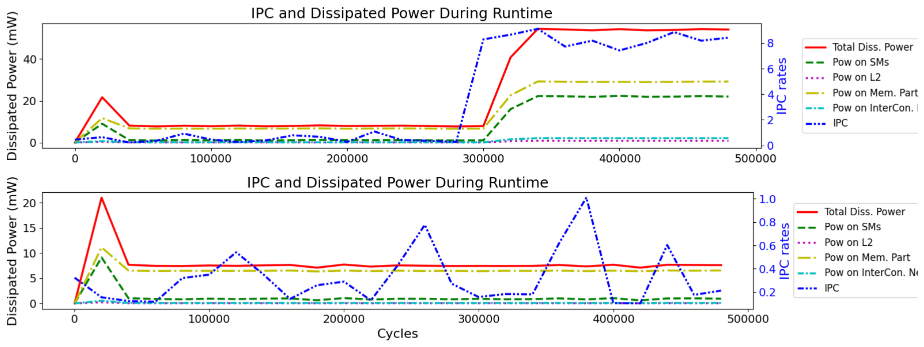


Fig. 17. IPC values for first 500K cycles for *softmax* kernel (matrix sizes with 1024 and 4096)

time for each component separately. GPPRMon further monitors warp-instruction issues/completions for each thread block, which provides an on-point analysis opportunity in a highly parallel execution. Hence, GPPRMon provides multi-functional performance and power tracking options together with various configurations, and it supports these features for all official GPU models described in Section 2.3.

TAU Performance System [39] is a performance profiling tool for hybrid parallel programs such as CUDA and OpenCL by intercepting the execution and inserting metric collection calls. After gathering the performance results, TAU integrates them with data through instrumentation to display a performance picture of the execution. Like Nsight Systems Tool [28], which provides performance traces for high-level CUDA functions such as *cudaMemcpy*, TAU does not address detailed hardware usage and performance during runtime as GPPRMon does. While TAU is insufficient to reveal the hardware's energy consumption, GPPRMon provides runtime power breakdown of GPU subcomponents during the execution.

Daisen [43] displays the overall occupancy on SM pipeline stages and memory components during the simulation. The authors aim to propose a performance-improving architecture by iterating an algorithm that benefits from the previously collected performance and hardware usage metrics. In other words, Daisen does not highlight the performance degrading points analytically. Instead, their approach addresses general bottlenecks, shares with the user, and tries to optimize performance in each iteration. Similar to [33], their approach offers more systematic performance optimization points on GPU executions. On the other hand, GPPRMon focuses on monitoring the execution at the PTX instruction level and relating the execution with the memory occupancy during execution. That is, GPPRMon offers a runtime analytical observation environment to evaluate both architecture and application inefficiencies. Moreover, while GPPRMon supports runtime power tracking and overall energy consumption metrics, Daisen does not provide them.

CHAMPVis [33] offers a web-supported architectural performance monitoring tool that provides a hierarchical analysis of trends and bottlenecks for varying applications. The tool aims to analyze performance by evaluating metrics from a system view and automatically generates predictive optimization speculations for the applications. Unlike GPPRMon, CHAMPVis does not employ any dissipated power tracking. GPPRMon yields detailed execution statistics, whereas CHAMPVis highlights the overall execution trends.

Francisco et al. [5] model a portion of the memory hierarchy of the AMD GPUs accurately by investigating the behavior of MSHRs, coalescing for vector (warp for our case) memory requests by extending Multi2Sim [45]. The authors find that the size and switching frequencies of MSHRs affect performance directly, especially for irregular workloads. Additionally, coalesced memory accesses, implemented in the simulator, reduce the repeated overheads of memory requests, significantly affecting performance for global memory accesses. GPPRMon offers more comprehensive evaluation opportunity besides the impact of MSHRs and memory request coalescing by being capable of runtime monitoring of both all memory hierarchy and SMs. Furthermore, GPPRMon extends relating the performance analysis options together with the dissipated power for performance/power trade-off evaluations at any execution scale.

Tanzima et al. [11] present an analysis and visualization framework (DASHING) targeting exascale computing consisting of multi-core architectures. The authors provide user interaction to compare varying configuration models by providing environment configuration options for analysis and visualization. Similarly, GPPRMon supports official GPU configurations of the GPGPU-Sim for multiple performance analyses and includes multi-functional metric collection and visualization perspectives depending on the user's demand. However, we obtain a more low-level performance analysis tool as described in *Temporal* and *Spatial View* Sections. In addition to runtime monitoring support, GPPRMon allows tracking the power dissipation across microarchitectural GPU components at runtime, which enables evaluating performance/power together, especially for energy-critical applications.

MemAxes [8] proposes an analysis framework for memory performance with various inspections on multi-core architectures. Its interface displays the analysis by obtaining performance metric samples from different simulations and mapping them into a single visual. In addition, the authors offer memory utilization-based clustering research among the benchmark applications. On the contrary, GPPRMon enables the analysis of GPU execution at runtime with performance and power dissipation features through *Spatial*, *Temporal*, and *General Views*.

To the best of our knowledge, GPPRMon differs from existing works by targeting hardware from both embedded and large-scale GPUs and providing runtime performance and power analysis opportunities. GPPRMon offers a detailed micro-architectural and power dissipation analysis for any GPU application. With observations through GPPRMon, developers can identify performance and power issues for both applications and architectures. For instance, data sparsity, irregular memory access behaviors, compute-bound behaviors, and the execution interval of these bottlenecks can be easily recognized via GPPRMon. Moreover, since GPPRMon supports runtime power dissipation together with performance monitoring, developers can analyze the energy impact of those performance issues and make their decisions. The studies conducting performance-energy tradeoff analysis [2, 38] and power capping strategies accordingly [49, 18] can potentially benefit from our fine-grained dynamic performance and power evaluations. We believe GPPRMon will be useful in fulfilling the micro-architectural performance and power analysis of GPUs for diverse application domains.

6. Conclusion

To conclude, we design and build a systematic runtime metric collection of instruction monitoring, performance, memory access, and power consumption metrics. Our tool provides a multi-perspective visualization framework that displays performance, execution statistics of the workload, occupancy of the memory hierarchy, and dissipated power results to conduct baseline analysis on GPUs at runtime. Since GPPRMon reliably reveals all the interactions between hardware and application at runtime, it potentially helps the researchers and software developers understand the dynamic behavior of the target GPU execution. While the researchers can propose architectural optimizations based on the detailed information, the software developers can deal with performance bottlenecks by analyzing our visualizations and fixing the target GPU code accordingly. Additionally, the low-power embedded system developers can utilize dynamic performance and power considerations to balance between two important design points.

We believe that GPPRMon will help to conduct baseline analysis for the literature concerning GPU performance and power dissipation and eliminate the need for additional inhouse efforts that involve real-time monitoring and profiling support. Since GPPRMon's metric collection and visualization components are independent, other microarchitectural metrics obtained during runtime from either GPUs or other simulators can be displayed by exploiting the GPPRMon visualizer. The integration of other simulators with GPPRMon results in a system capable of runtime execution, memory statistics, and power dissipation tracker among all the possible GPUs. Alternatively, extending the runtime visualization by deepening the scope of the runtime metric collection and visualization phases of GPPRMon is another future direction.

Acknowledgments. This work was supported by the Scientific and Technological Research Council of Turkey (TÜBİTAK), Grant No: 122E395. This work is partially supported by CERCIRAS COST Action CA19135 funded by COST Association.

References

1. Ariel, A., Fung, W.W.L., Turner, A.E., Aamodt, T.M.: Visualizing complex dynamics in many-core accelerator architectures. In: 2010 IEEE International Symposium on Performance Analysis of Systems & Software (ISPASS). pp. 164–174 (2010)
2. Aslan, B., Yilmazer-Metin, A.: A study on power and energy measurement of nvidia jetson embedded gpus using built-in sensor. In: 2022 7th International Conference on Computer Science and Engineering (UBMK). pp. 1–6 (2022)
3. del Barrio, V., Gonzalez, C., Roca, J., Fernandez, A., E, E.: Attila: a cycle-level execution-driven simulator for modern gpu architectures. In: 2006 IEEE International Symposium on Performance Analysis of Systems and Software. pp. 231–241 (2006)
4. Becker, P.H.E., Arnau, J.M., González, A.: Demystifying power and performance bottlenecks in autonomous driving systems. In: 2020 IEEE International Symposium on Workload Characterization (IISWC). pp. 205–215 (2020)
5. Candel, F., Petit, S., Sahuquillo, J., Duato, J.: Accurately modeling the gpu memory subsystem. In: 2015 International Conference on High Performance Computing & Simulation (HPCS). pp. 179–186 (2015)
6. Chen, X., Chang, L.W., Rodrigues, C.I., Lv, J., Wang, Z., Hwu, W.M.: Adaptive cache management for energy-efficient gpu computing. In: 2014 47th Annual IEEE/ACM International Symposium on Microarchitecture. pp. 343–355 (2014)

7. Collange, C., Daumas, M., Defour, D., Parello, D.: Barra: A parallel functional simulator for gpgpu. In: 2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems. pp. 351–360 (2010)
8. Giménez, A., Gamblin, T., Jusufi, I., Bhatele, A., Schulz, M., Bremer, P.T., Hamann, B.: Memaxes: Visualization and analytics for characterizing complex memory performance behaviors. IEEE Transactions on Visualization and Computer Graphics 24(7), 2180–2193 (2018)
9. Guerreiro, J., Ilic, A., Roma, N., Tomás, P.: Dvfs-aware application classification to improve gpgpus energy efficiency. Parallel Computing 83, 93–117 (2019), https://www.sciencedirect.com/science/article/pii/S0167819118300243
10. Hong, J., Cho, S., Kim, G.: Overcoming memory capacity wall of gpus with heterogeneous memory stack. IEEE Computer Architecture Letters 21(2), 61–64 (2022)
11. Islam, T., Ayala, A., Jensen, Q., Ibrahim, K.: Toward a programmable analysis and visualization framework for interactive performance analytics. In: 2019 IEEE/ACM International Workshop on Programming and Performance Visualization Tools (ProTools). pp. 70–77 (2019)
12. Jain, P., Jain, A., Nrusimha, A., Gholami, A., Abbeel, P., Keutzer, K., Stoica, I., Gonzalez, J.E.: Checkmate: Breaking the memory wall with optimal tensor rematerialization. CoRR abs/1910.02653 (2019), http://arxiv.org/abs/1910.02653
13. Jog, A., Kayiran, O., Nachiappan, N.C., Mishra, A.K., Kandemir, M.T., Mutlu, O., Iyer, R.R., Das, C.R.: OWL: cooperative thread array aware scheduling techniques for improving GPGPU performance. In: Sarkar, V., Bodík, R. (eds.) Architectural Support for Programming Languages and Operating Systems, ASPLOS 2013, Houston, TX, USA, March 16-20, 2013. pp. 395–406. ACM (2013), https://doi.org/10.1145/2451116.2451158
14. Kandiah, V., Peverelle, S., Khairy, M., Pan, J., Manjunath, A., Rogers, T.G., Aamodt, T.M., Hardavellas, N.: Accelwattch: A power modeling framework for modern gpus. In: MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture. p. 738–753. MICRO '21, Association for Computing Machinery, New York, NY, USA (2021), https://doi.org/10.1145/3466752.3480063
15. Khairy, M., Shen, Z., Aamodt, T.M., Rogers, T.G.: Accel-sim: An extensible simulation framework for validated gpu modeling. In: 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA). pp. 473–486 (2020)
16. Koo, G., Oh, Y., Ro, W.W., Annavaram, M.: Access pattern-aware cache management for improving data utilization in gpu. In: 2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA). pp. 307–319 (2017)
17. Krzywaniak, A., Czarnul, P., Proficz, J.: Gpu power capping for energy-performance trade-offs in training of deep convolutional neural networks for image recognition. In: Computational Science – ICCS 2022. pp. 667–681. Springer International Publishing, Cham (2022)
18. Krzywaniak, A., Czarnul, P., Proficz, J.: Dynamic gpu power capping with online performance tracing for energy efficient gpu computing using depo tool. Future Generation Computer Systems 145, 396–414 (2023), https://www.sciencedirect.com/science/article/pii/S0167739X23001267
19. Leskovec, J., Lang, K., Dasgupta, A., Mahoney, M.: Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. Internet Mathematics 6 (11 2008)
20. Lew, J., Shah, D.A., Pati, S., Cattell, S., Zhang, M., Sandhupatla, A., Ng, C., Goli, N., Sinclair, M.D., Rogers, T.G., Aamodt, T.M.: Analyzing machine learning workloads using a detailed gpu simulator. In: 2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS). pp. 151–152 (2019)
21. Li, S., Ahn, J.H., Strong, R.D., Brockman, J.B., Tullsen, D.M., Jouppi, N.P.: Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In: 2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). pp. 469–480 (2009)

22. Mittal, S., Vetter, J.S.: A survey of methods for analyzing and improving gpu energy efficiency. ACM Comput. Surv. 47(2) (aug 2014), https://doi.org/10.1145/2636342
23. NVIDIA: Quadro gv100 data sheet (2018), https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/productspage/quadro/quadro-desktop/quadro-volta-gv100-a4-nvidia-704619-r3-web.pdf
24. NVIDIA: Volta architecture white paper (March 2018), https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf
25. NVIDIA: Jetson agx xavier and the new era of autonomous machines (2019), https://info.nvidia.com/rs/156-OFN-742/images/Jetson_AGX_Xavier_New_Era_Autonomous_Machines.pdf
26. NVIDIA: Cuda toolkit documentation (Jan 2023), https://docs.nvidia.com/cuda
27. NVIDIA: Nsight compute kernel profiling guide (2024), https://docs.nvidia.com/nsight-compute/ProfilingGuide/index.html
28. NVIDIA: Nsight systems profiling guide (2024), https://developer.nvidia.com/nsight-systems
29. NVIDIA: Nvidia cuda compiler driver nvcc (2024), https://docs.nvidia.com/cuda/cuda-compiler-driver-nvcc/
30. NVIDIA: Parallel thread execution isa version 8.4 (2024), https://docs.nvidia.com/cuda/parallel-thread-execution/index.html
31. NVIDIA: System management interface (2024), https://developer.nvidia.com/system-management-interface
32. O'Neil, M.A., Burtscher, M.: Microarchitectural performance characterization of irregular gpu kernels. In: 2014 IEEE International Symposium on Workload Characterization (IISWC). pp. 130–139 (2014)
33. Pentecost, L., Gupta, U., Ngan, E., Beyer, J., Wei, G.Y., Brooks, D., Behrisch, M.: Champvis: Comparative hierarchical analysis of microarchitectural performance. In: 2019 IEEE/ACM International Workshop on Programming and Performance Visualization Tools (ProTools). pp. 55–61 (2019)
34. Power, J., Hestness, J., Orr, M.S., Hill, M.D., Wood, D.A.: gem5-gpu: A heterogeneous cpu-gpu simulator. IEEE Computer Architecture Letters 14(1), 34–36 (2015)
35. PowerUP, T.: V100 tech powerup (2018), https://www.techpowerup.com/gpu-specs/tesla-v100-pcie-16-gb.c2957
36. Rhu, M., Sullivan, M., Leng, J., Erez, M.: A locality-aware memory hierarchy for energy-efficient gpu architectures. In: Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture. p. 86–98. MICRO-46, Association for Computing Machinery, New York, NY, USA (2013), https://doi.org/10.1145/2540708.2540717
37. Rodriguez, I., Kosmidis, L., Lachaize, J., Notebaert, O., Steenari, D.: Gpu4s bench: Design and implementation of an open gpu benchmarking suite for space on-board processing. Universitat Politecnica de Catalunya (2019)
38. Sezgin, Y., Öz, I.: Performance-reliability tradeoff analysis for safety-critical embedded systems with gpus. In: Ulusal Yüksek Başarımlı Hesaplama Konferansı (BAŞARIM) (2024), https://indico.truba.gov.tr/event/140/attachments/310/642/BASARIM2024-BildiriKitabi.pdf
39. Shende, S.S., Malony, A.D.: The tau parallel performance system. Int. J. High Perform. Comput. Appl. 20(2), 287–311 (may 2006), https://doi.org/10.1177/1094342006064482
40. Shi, X., Zheng, Z., Zhou, Y., Jin, H., He, L., Liu, B., Hua, Q.S.: Graph processing on gpus: A survey. ACM Comput. Surv. 50(6) (jan 2018), https://doi.org/10.1145/3128571
41. Strohmaier, E., Dongarra, J., Simon, H., Meuer, M.: Highlights - november 2022 (2022), https://www.top500.org/lists/top500/2022/11/highs/
42. Sun, Y., Mukherjee, S., Baruah, T., Dong, S., Gutierrez, J., Mohan, P., Kaeli, D.: Evaluating performance tradeoffs on the radeon open compute platform. In: 2018 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS). pp. 209–218 (2018)

43. Sun, Y., Zhang, Y., Mosallaei, A., Shah, M.D., Dunne, C., Kaeli, D.R.: Daisen: A framework for visualizing detailed GPU execution. Comput. Graph. Forum 40(3), 239–250 (2021), https://doi.org/10.1111/cgf.14303
44. Topçu, B., Öz, I.: Gpprmon: Gpu runtime memory performance and power monitoring tool. In: Euro-Par 2023: Parallel Processing Workshops. pp. 17–29. Springer Nature Switzerland (2024)
45. Ubal, R., Jang, B., Mistry, P., Schaa, D., Kaeli, D.: Multi2sim: A simulation framework for cpu-gpu computing. In: 2012 21st International Conference on Parallel Architectures and Compilation Techniques (PACT). pp. 335–344 (2012)
46. Vijaykumar, N., Ebrahimi, E., Hsieh, K., Gibbons, P.B., Mutlu, O.: The locality descriptor: A holistic cross-layer abstraction to express data locality in gpus. In: 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA). pp. 829–842 (2018)
47. Wang, Y., Pan, Y., Davidson, A., Wu, Y., Yang, C., Wang, L., Osama, M., Yuan, C., Liu, W., Riffel, A.T., Owens, J.D.: Gunrock: Gpu graph analytics. ACM Trans. Parallel Comput. 4(1) (aug 2017), https://doi.org/10.1145/3108140
48. Xu, Z., Chen, X., Shen, J., Zhang, Y., Chen, C., Yang, C.: Gardenia: A graph processing benchmark suite for next-generation accelerators. ACM Journal on Emerging Technologies in Computing Systems 15(1) (jan 2019), https://doi.org/10.1145/3283450
49. Zhao, D., Samsi, S., McDonald, J., Li, B., Bestor, D., Jones, M., Tiwari, D., Gadepally, V.: Sustainable supercomputing for ai: Gpu power capping at hpc scale. In: Proceedings of the 2023 ACM Symposium on Cloud Computing. p. 588–596. SoCC '23, Association for Computing Machinery, New York, NY, USA (2023), https://doi.org/10.1145/3620678.3624793
50. Zhao, X., Adileh, A., Yu, Z., Wang, Z., Jaleel, A., Eeckhout, L.: Adaptive memory-side last-level gpu caching. In: 2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA). pp. 411–423 (2019)

Burak Topçu is a PhD student in the Department of Computer Science and Engineering at Pennsylvania State University. He received a B.Sc. degree from Middle East Technical University and an M.Sc. degree from Izmir Institute of Technology.

Deniz Karabacak is an undergraduate student in the Electrical and Electronics Engineering Department at Izmir Institute of Technology.

Işıl Öz is an assistant professor in the Computer Engineering Department at Izmir Institute of Technology. Her research interests include computer architecture, multicore systems, heterogeneous systems, and fault-tolerant computing.

Received: July 22, 2024; Accepted: November 05, 2024.

Computer Science and Information Systems 22(2):563–590 https://doi.org/10.2298/CSIS240722023K

Comparison and Analysis of Software and Hardware Energy Measurement Methods for a CPU+GPU System and Selected Parallel Applications

Grzegorz Koszczał, Mariusz Matuszek, and Paweł Czarnul

Faculty of Electronics, Telecommunications and Informatics
Gdańsk University of Technology
Narutowicza 11/12
80-233 Gdańsk, Poland
pczarnul@eti.pg.edu.pl

Abstract. In this paper authors extend upon their previous research on power-capped optimization of performance-energy metrics of deep neural networks training workloads. A professional power meter Yokogawa WT-310E is used, as well as Intel RAPL and Nvidia NVML interfaces, to examine power consumption of a much more comprehensive set of multi-GPU and multi-CPU workloads, including: selected kernels from NAS Parallel Benchmarks for CPUs and GPUs as well as Horovod-Python Xception deep neural network training using several GPUs. A comparison and discussion of results obtained by both power measurement methods has been performed using 2 systems, one with 2 Intel Xeon CPUs and 8 Nvidia Quadro RTX 6000 GPUs and the second 2 Intel Xeon CPUs and 4 Nvidia Quadro RTX 5000 GPUs. We compared power consumption between hardware and software interfaces for CPU, GPU and mixed CPU+GPU workload configurations, using 1-40 threads for the CPUs and 1-8 GPUs.

Keywords: high performance computing, performance-energy optimization, energy measurements, measurement accuracy, parallel benchmarks.

1. Introduction

In recent years an increased awareness of the need for energy conservation can be observed, together with ever-increasing demand for high performance computing (HPC), either in a traditional or cloud-presented form. These two factors combined led to a series of research on performance-energy optimizations in high performance computing environments, as well as introducing dedicated power measurement interfaces: Intel RAPL[1] and Nvidia NVML[2].

In this paper we use both a Yokogawa WT-310E professional power meter [41] and software interfaces by Intel and Nvidia to examine reported power draw under a comprehensive set of multi-GPU, multi-CPU and mixed workloads, in order to gain a better insight into capabilities and limitations of those software interfaces.

[1] Intel Running Average Power Limit (RAPL) is a set of interfaces for reporting the accumulated energy consumption of various power domains [17].

[2] Nvidia Management Library (NVML) is a C-based API that allows monitoring and managing various states of the NVIDIA GPU devices, including their power draw [32].

This paper is a very significantly extended version of conference paper [25]. While the original work focused on power-capped optimization of performance-energy metrics of multi-GPU training of deep neural networks using a hardware power meter, this paper focuses on a thorough comparison of software and hardware power/energy measurement methods using the aforementioned application (Xception DNN training) and a set of selected NAS Parallel Benchmarks, run on CPUs, GPUs and CPUs+GPUs configurations.

The paper is organized in a standard pattern. We begin by highlighting relevant research papers in this field. Next we present our motivation for research and list our main contributions. Then a description of our measurement methodology is given, followed by reasoning behind our choice of testbed applications. A large part of this paper is then dedicated to a detailed description of our experiments and discussion of results. We finish the paper with a summary of results and short discussion on possible future work and an appendix, containing detailed measurements including power values from individual computing devices and entire nodes, execution times and standard deviations for the measurements.

2. Related Work

The related work can be broadly assigned to two research categories. One is examination and validation of RAPL and NVML power measurement interfaces themselves. The second one is using measurements obtained from said interfaces as inputs for some energy/performance optimization scheme. Additionally, there are papers dedicated to research on the accuracy and quality of standard benchmark patterns. Papers [30] and [6] present NAS Parallel Benchmarks and their implementations, while papers [3] and [4] discuss efficient NAS Parallel benchmarks on GPUs.

Hähnel et al. [14] report on using Intel RAPL energy sensors to measure energy consumption of short code paths, with execution times substantially shorter than RAPL update interval. The authors describe operating system level modifications made to ensure code execution synchronous with RAPL register updates.

Lang and Rünger [28] propose a statistical method for generating high resolution power profiles on Nvidia GPUs using standard (lower) resolution power measurements returned by NVML.

A power consumption model focused on rendering graphics of varying complexity on a mobile GPU is proposed by Vatjus-Anttila et al. [39]. The model is based on graphics primitives, such as triangles, render batches and texels, and thus is intended to be hardware agnostic. The proposed approach allows to predict the complexity of any given 3D scene at a content production phase. The authors verify their model with measurements on a real-world content and hardware and report prediction errors in range of 0.3% to 3.2%.

A quantitative evaluation of the Intel's RAPL power control system is presented by Zhang and Hoffmann [42]. The authors evaluate the RAPL system by setting target power limits and running a set of benchmarks, quantifying the results using metrics of accuracy, stability, settling time, overshoot and efficiency.

Desrochers et al. [11] conducted detailed analysis of RAPL measurements (using the perf event interface) concerning the CPU and DRAM versus system wide numbers from a WattsUpPro? device. The testbed system included 3 machines with 2 desktop and 1 server Haswell CPUs. Three different types of DDR3 DRAM as well as two types of

DDR4 DRAM were tested. The authors concluded that, in general, usually RAPL measurements follow the total power value trends, typically by a constant offset. In general measurements matched within 20% with following relative phase behavior. Additionally, in terms of memory, results matched best when the DRAM was heavily utilized, but were not accurate where the system was idle or memory was used by an integrated GPU. The authors also noted that the Haswell server machines returned more accurate results from actual power measurements.

Lucas and Juurlink [29] examine the power consumption dependency of arithmetic-logic units (ALU's) in modern GPUs on data values being processed. They show large power consumption differences between the same kernel processing different data. Use of microbenchmarks to characterize power consumption of GPU functional units is discussed and obtained data is used to propose a simple and fast model of the power consumption of functional units, in order to improve accuracy of power consumption models.

Ferro et al. [13] use internal GPU power sensors to examine power profiles of two Nvidia GPUs under both artificial (benchmark) and real application loads. Cards from two different generations are measured and a comparison of accuracy between generations is made. Power consumption of a whole node is also monitored using IPMI interface and tool.

Ikram et al. [15] propose an experimental methodology for evaluating power and energy consumption of programs executing on Nvidia Kepler GPUs. The methodology is applied to two common HPC programs: a matrix multiplication and a parallel sorting. The authors conclude, that their methodology can be applied to any program executing on Nvidia Kepler GPU, to obtain measurements of peak and average power, energy and kernel runtime.

Khan et al. [23] performed detailed analysis of RAPL accuracy and usefulness for power measurements. They used both customized benchmarks as well as Stream, Stress-ng and ParFullCMS applications as well as two power measurement datasets from the Taito – a supercomputing system of the Finnish Center of Scientific Computing. They determined that measurements using RAPL are very highly highly correlated (≈ 0.99) with AC power, with a high sampling rate $\approx$ 1ms that even allows to distinguish various phases of an application and negligible overhead less than 1% for standalone systems and less than 2.5% for Amazon EC2. They concluded that RAPL can be used for measuring energy consumption of servers without power meters. Selected, desired features that would be useful were identified, including: reduction of the delay of reading RAPL MSRs through hypervisors and more resolution, i.e., per core readings, not only per package.

Sen et al. [35] present a quality assessment methodology of GPU power profiling mechanisms. Using the proposed methodology the authors assess the quality of four profiling techniques: NVML via Allinea MAP, NVML via direct reads, PowerInsight (PI) via vendor-provided drivers and PI via Allinea MAP. In addition the authors discuss the influence of moving-average filters on the slow variation of some of measured power profiles.

Paniego et al. [33] focus on monitoring and analyzing energy consumption in HPC environment for a given application and architecture. A matrix multiplication application on a shared-memory architecture is used to compare values reported by Intel RAPL with physical measurements obtained through the processor power source. The authors report that, for the application considered, there is an error of up to 22% between the average CPU power and values predicted by RAPL.

Fahad et al. [12] investigated accuracy of on-chip power sensors and predictive models versus an external hardware power meter for measurement of dynamic energy of applications. They ran two scientific applications, matrix-matrix multiplication and 2D fast Fourier transform, for various data sizes, on three Intel multi-core CPUs, two Nvidia GPUs and an Intel Xeon Phi device and concluded that the average error using on-chip power sensors can be as high as 73% and using predictive models with performance monitoring counters can be as high as 32% for Haswell and Skylake CPUs, for dynamic energy profiles.

Kavanagh and Djemame [20] use RAPL and IPMI interfaces to come up with tuned models for estimation of host-level power consumption, for use in Cloud and High Performance Computing platforms. The methodology of models tuning and calibration, as well as mitigating errors present in both interfaces is discussed in detail.

Ilsche [16] analyzed, in particular, the accuracy of RAPL measurements, for various architectures of Intel CPUS, specifically: Sandy Bridge, Haswell and Skylake. For SandyBridge, after testing several workloads, for the package domain correlation between RAPL and reference power has been found to be weak with RAPL values being higher or lower, depending on the workload. For DRAM, close correlation was observed for higher power consumption while visible discrepancy was observed for smaller values. This suggests that usage of RAPL is limited in that generation as a replacement for real measurements, according to the author. In that generation RAPL used an internal, architectural model with a set of events and consideration of weights for prediction of power consumption. From Haswell, an implementation based on physical measurements was introduced. The author also evaluated RAPL for the Skylake generation. The accuracy of measurements for that generation was shown to be considerably better, with relative discrepancy for package + DRAM max. at 3.8% and 3.3% without idle measurement point (but with much higher discrepancies for separate package and DRAM components).

Arafa et al. [2] present a detailed measurement of energy consumption of different instructions that can be executed on Nvidia GPGPUs. Three different techniques to read on-chip power sensors are being used and a comparison of these techniques is made. Additionally, obtained measurements are verified against an external, custom-designed, hardware power measuring device.

Aslan and Yilmazer-Metin [5] present a study on power and energy measurement in Nvidia Jetson embedded GPUs, by validating and extending the methodology presented by Burtscher et al. [8], originally developed for measurement of GPU power using the K20 built-in sensor.

Shahid et al. [36] investigated various energy predictive models for multi-core CPUs, performing tests for two systems with Haswell and Skylake CPUs and a variety of benchmarks including: NPB kernels, MKL FFT, HPCG, MKL DGEMM, stress and naive matrix-matrix and matrix-vector multiplication. They have determined, having analyzed the prediction accuracy of linear energy models based on utilization variables only, PMCs only and both utilization variables and PMCs, that the best models with both utilization variables and PMCs resulted in 3.6 and 2.6 times better average prediction accuracy compared to the models based on utilization variables only and PMCs only.

Krzywaniak et al. [26,27] analyzed application of power capping for multi-core CPUs and GPUs, in the context of performance-energy optimization considering metrics such as Energy Delay Product (EDP), Energy Delay Sum (EDS). For a given parallel application and a computing device, the goal was to find such a power cap so that the value of a given metric was optimized, compared to the default power cap. Since this process was performed dynamically at runtime by the proposed DEPO tool, available in versions for the CPU and the GPU, both performance and power/energy were measured automatically, even without the need for code instrumentation. For a given power cap, in the case of the CPU, application progress is measured using an instruction counter and energy using Intel RAPL. For a code running on a GPU, application progress is measured using a kernel invocation counter while power is measured using Nvidia NVML which allows computation of energy. Measurements are taken in an adjustable time window that should be short enough for quick browsing of a range of available power caps and sufficiently long for the measurements to be stable and representative. DEPO can browse the available range of adjustable power caps using one of two algorithms: Linear Search (LS) and Golden Section Search (GSS). As a result, the authors concluded that for the purpose of the stated optimization, the accuracy of Intel RAPL and Nvidia NVML was sufficient, as demonstrated by Krzywaniak et al. [26] (RAPL compared to HPE Metered 3Ph 22 kVA/60309 5-wire 32 A/230 V Outlets (30) C13 (3) C19/Vertical INTL PDU (D9N56A) with ±1% or better accuracy in power monitoring) and [27] (NVML compared to power meter WT310) respectively.

A very low-level study of energy cost associated with dynamic branch prediction in an Intel CPU is given by Alqurashi and Al-Hashimi [1]. The authors use the RAPL interface for reporting the CPU power and energy, while using known micro-benchmarks under various run conditions to explore potential pitfalls in the measurement interface.

Departing from the prevailing Intel RAPL and Nvidia NVML tune, Tröpgen et al. [38] show an evaluation of the energy measurement present in the IBM POWER9 on-chip controller (OCC). The authors provide a detailed description and in-depth evaluation of OCC-provided power measurements for several power domains, confronting the results with externally measured data.

Yang et al. [40] investigated details of power/energy measurement using Nvidia-smi. They proposed a suite of micro-benchmarks to benchmark profiles using Nvidia-smi for power readings and have evaluated for over 70 different GPUs from several generations. For the steady state error evaluation, in the majority of the cases the error was within +/-5%. Additionally, they determined that for H100 and A100, Nvidia-smi only reports the average of the past 25ms every 100ms, while for the previous Volta and Pascal generations 10/20ms periods were reported. The authors proposed to incorporate controlled delays between repetitions to exposing various segments of an application to the identified measurement window.

Shalavi et al. [37] study the accurate calibration of power measurements from Nvidia Jetson devices. The authors propose and utilise a regression model to map sensor measurements to real accurate power readings obtained from external hardware. The authors found, that internal sensors in Jetson devices underestimated the power draw by up to 50% and, by applying calibration, were able to reduce the error to within $\pm 3\%$.

A very interesting and thorough experimental comparison of software-based power meters was presented by Jay et al. [19], focusing on both CPU and GPU workloads. The authors discussed various methods to measure energy in computer systems, including: power meters, intra-node devices, hardware sensors (with respective software interfaces such as Intel RAPL and Nvidia NVML), as well as power and energy modeling (including using e.g. a combination of RAPL and hardware performance counter events). Several software-based power meters are discussed and compared, including: Carbon Tracker, Code Carbon, Energy Scope, Experiment Impact Tracker, Perf, Power API, Green Algorithms, ML CO2 Impact, and Scaphandre. Furthermore, experimental evaluation was conducted using a cluster with Nvidia DGX-1 nodes, each with 2 Intel Xeon (Broadwell) CPUs, 8 Nvidia Tesla V100 GPUs and 512 GB RAM, with an Omegawatt external power meter (sampling frequency of 1Hz) and node's BMC with a sampling frequency of 0.2Hz. Selected NAS benchmark kernels were used: EP, LU and MG (different sizes for CPUs and GPUs). Power profiles over time were analyzed from Energy Scope, Scaphandre, Perf, PowerAPI, BMC, and the power meter for both the CPU and GPU. In general, the Pearson correlation coefficient between the tools and the power meter was approx. 0.95 with the highest value for EnergyScope 0.972 (sampling frequency of 2Hz). What is very interesting in the context of this work is that the offset between the software power meters' and the external power meter' values is not constant and, according to the authors, shall be studied for various architectures/computing node. Energy overheads of the tools were evaluated as of approx. 1%. Additionally, the authors tested several benchmarks running in parallel, also benchmarks of various types, i.e., compute and memory intensive, with PowerAPI and Scaphandre giving mostly coherent results but some differences occurred as well. One interesting observation from the results was that after GPU workloads, GPUs would still take some 20-30 seconds to return to idle load, apparently due to engaged fans etc.

Raffin and Trystram [34] perform a critical analysis of Intel RAPL energy measurement operations in order to provide RAPL users with best access practices. Qualitative highlights of differences between measurement mechanisms are given, including evaluation of overheads for various mechanisms present. The paper is focused on a comparative analysis of the measurement mechanisms and on experimental evaluation of performance and energy measurements.

3. Motivations and Contribution

As energy-aware high performance computing has gained much attention [10,24], there is a constant need for assessment of accuracy of various power and energy measurement methods. This stems from the fact that methods such as Intel RAPL and Nvidia NVML might be expected to be improved over time and also because of the fact that many system configurations differ considerably in terms of setups such as numbers of CPUs, GPUs but also density and additional devices that might be measured by hardware but not grasped by software APIs, notably fans etc.

Our direct motivation for this research is following up on our previous research on power-capped optimization of performance-energy metrics of multi-GPU training of deep neural networks [25]. While in the latter work, we took power measurements under power capping conditions using a professional hardware power meter Yokogawa WT-310E [41] and used DNN training workloads only, in this paper we focuse on comparing power readings from Yokogawa and Intel RAPL and Nvidia NVML. For the comparison to be meaningful, we broaden the scope of workloads with NPB Suite to better reflect the diversity of contemporary applications. Additionally, we want to explore potential impact of possible power supply configurations on such measurements.

Our contribution is a comparison of power consumption readings between software and hardware measurement methods, for a set of workloads representative of HPC applications, for a range of CPU and GPU loads, up to 2 multi-core CPUs and 8 GPUs. We also expose and discuss power consumption overheads which are not visible in the readings from software interfaces like Intel RAPL and Nvidia NVML.

4. Measurement Methodology

The methodology adopted in this paper assumes that we compare average power taken by a system running selected, representative benchmarks, using both hardware (power meter) and software based measurement methods (through Intel RAPL and Nvidia NVML) which are available and widely used by other researchers [19].

The ground truth for our measurements is provided by Yokogawa WT310E – an external power meter that is a digital power analyzer that provides extremely low current measurement capability down to 50 micro-Amps, and a maximum of up to 26 Amps RMS. This device follows standards and certificates such as Energy Star, SPECpower and IEC62301/EN50564 testing. This model belongs to WT300E's family of devices that offer a wide range of functions and enhanced specifications, allowing handling of all the measurement applications from low frequency to high frequency inverters using a single power meter. The WT300E series with a fast update rate of 100ms. The basic accuracy for all input ranges is 0.1% rdg + 0.05% rng (50/60Hz) and DC 0.1% rdg + 0.2% rng. In order to obtain readings from the Yokogawa WT310E power meter connected to a machine, a special software has been written – Yokotool [7]. Yokotool is a command-line tool for controlling Yokogawa power meters in Linux. The tool is written in Python and comes with the 'yokolibs.PowerMeter' module which can be used from Python scripts.

For software-based measurement APIs, we used Intel RAPL for obtaining energy and then power from the CPU(s) and the DRAM domain as well as Nvidia NVML for obtaining power taken by the GPU(s). Energy measurements from RAPL are accessible by reading proper files exposed in a file system. Power values are reported by NVML by either using the `nvidia-smi` command-line tool or calling the `nvmlDeviceGetPowerUsage` library function.

While Yokogawa allows us to measure the total power of the whole system, the combined readings from RAPL and NVML are not able to account for additional system devices, including disks and especially cooling, including fans. By analyzing the difference between the Yokogawa and combined RAPL+NVML measurements, under various configurations (CPU cores and GPUs used), for various applications, we can conclude how the difference changes with the configurations, applications and systems.

As the measurement frequency of the Yokogawa device is 10Hz, it was the frequency that we set for all the APIs, i.e., Yokogawa WT301E, Intel RAPL and Nvidia NVML.

Specifically, the following results will include computation of the offset (δ) as the difference between the subtraction of active node power draw (Yokogawa) (θ_Y) and the sum of active CPUs (RAPL) power draw (θ_R) and GPUs (NVML) power draw (θ_N) [W], and subtraction of idle node (Yokogawa) power draw (θ_{Y_i}) and the sum of idle CPUs and

GPUs (RAPL+NVML) power draw [W] (θ_{R_i} and θ_{N_i}, respectively), computed for each configuration:

$$\begin{aligned}\delta &= (\theta_Y - (\theta_R + \theta_N)) - (\theta_{Y_i} - (\theta_{R_i} + \theta_{N_i})) \\ &= (\theta_Y - \theta_{Y_i}) - ((\theta_R + \theta_N) - (\theta_{R_i} + \theta_{N_i}))\end{aligned} \tag{1}$$

5. Testbed Applications

In order to obtain results that are representative of a wide range of applications, it is best to rely on performing experiments using well established benchmarks. In our case, it is especially important that we target CPU, GPU and mixed CPU+GPU workloads, for the purpose of comparison of power/energy measurement methods. Specific requirements include: ability to run in parallel on various numbers of logical processors, run on one or more GPUs and being able to execute for various input data sizes so that configurations with reasonable compute to communication/synchronization time ratios can be selected so that parallelization is efficient, actual speed-ups are obtained [9]. We have decided to use the well-established NAS Parallel Benchmark Suite as well as loads generated by Xception DNN training. The NPB Suite contains a series of kernels including: IS – Integer Sort (random memory access), EP – Embarassingly Parallel, CG – Conjugate Gradient (irregular memory access and communication), MG – Multi-Grid on a sequence of meshes, FT – Discrete 3D fast Fourier Transform (all-to-all communication) as well as applications: BT – Block Tri-diagonal solver, SP – Scalar Penta-diagonal solver, LU – Lower-Upper Gauss-Seidel solver. These benchmarks are available in various sizes, in particular: classes A, B, C – standard test problems (roughly 4 times size increase from each of the previous classes) as well as classes D, E, F – large test problems (roughly 16 times size increase from each of the previous classes). In terms of the implementations tested, we used the following that satisfied the aforementioned criteria to put load on the CPU(s) and GPU(s): NAS Parallel Benchmarks (C++ with OMP) [30], NAS Parallel Benchmarks (Fortran with MPI) [6], NAS Parallel Benchmarks (CUDA) [3,4]. In addition to the NPB Suite, a custom deep learning model based on Xceptionnet with MPI communication was used. This model was tested previously with power capping for performance-energy optimization under power capping [25]. The rationale and methodology for selection of particular configurations is discussed in detail in Section 6.2.

It should be noted how sizes of the benchmarks are set in the following experiments, which stems from actual implementations for CPUs and GPUs. Specifically, the benchmarks for the CPU scale with a specific number of threads set which results in reduced execution times for larger numbers of threads. On the other hand, the compiled benchmarks for the GPU are run, by default, on a single GPU. For runs that use more than one GPU, we run the same benchmark independently on a given number of GPUs, which results in essentially the same execution time of the test, regardless of the number of GPUs used. In the case of mixed CPU+GPU benchmarks, we conducted a given test until the end of the shortest (out of the CPU and GPU) benchmarks.

6. Experiments

6.1. Testbed Systems

For the following tests, we used two systems with multicore CPUs and GPUs, with the following specifications:

vinnana : 2 x Intel(R) Xeon(R) Silver 4210 CPU (TDP 85W), 2.20GHz, for a total of 20 physical cores and 40 logical processors; 384 GB RAM, DDR4, 2400 MHz; 4 x Nvidia Quadro RTX 5000 16GB (TDP 230W);

sanna : 2 x Intel(R) Xeon(R) Silver 4210 CPU (TDP 85W), 2.20GHz, for a total of 20 physical cores and 40 logical processors; 384 GB RAM, DDR4, 2400 MHz; 8 x Nvidia Quadro RTX 6000 24GB (TDP 260W).

Each system uses an Inspur YZMB01130107 motherboard as well as 4 × Delta Electronics DPS-2200AB-2 PSUs and runs Linux Ubuntu 22.04 LTS operating system. Use of two very similar systems allowed us to achieve two objectives:

1. validate our findings (in the load ranges common to the two systems, to the extent possible with our experiment setup),
2. investigate the power consumption overhead patterns between less and more densely GPU equipped systems, which was made possible by otherwise identical configurations of the two systems. The only difference between the two systems was the number and models of the installed GPUs.

6.2. Tested Configurations

Tested configurations correspond to various parallelization levels of application runs, interesting from the point of view of taking advantages of the CPUs' numbers of cores and numbers of GPUs, installed in the two testbed systems. Consequently, we varied the number of CPU threads between 2 and 32 threads, considering powers of 2 for selected simulations which is a typical approach in parallel computing benchmarking. For others, we set the number of threads between 1 and 20 for 1 CPU as well as between 2 and 40 for 2 physical CPUs which is adjusted to the actual CPU models and core and logical processor counts described in Section 6.1. For the GPU tests, we varied the number of GPUs used, as available in a given system: 1, 2 and 4 GPUs for vinnana and 1, 2, 4 and 8 GPUs in sanna. For mixed CPUs+GPUs tests, we combined increasing numbers of threads and GPUs from the aforementioned configurations.

In order to provide a larger variety of tests on similar machines (different in the GPU configuration) we have decided to conduct tests of various benchmarks/implementations on the two machines, as follows:

vinnana : CPUs – MPI-Fortran[3], GPUs – Horovod-Python, mixed – MPI-Fortran + Horovod-Python;

sanna : CPUs – OMP-CPP, GPUs – OMP-CUDA, mixed – OMP-CPP + OMP-CUDA.

[3] note that throughout the paper, in the case of MPI-Fortran workloads, we refer to computational threads.

In order to select configurations for testing, we adopted the following assumptions. Considering the sampling frequency of Yokogawa WT-310E of 10Hz, we assumed that each test should last at least 20 seconds (in fact guaranteeing 199 probes). We performed initial tests in order to select benchmarks and corresponding classes (sizes) to satisfy this criterion. The following benchmarks and configurations were run in this initial phase on particular systems, each of which was run 3 times and average values were recorded:

Implementation: OMP-CPP: benchmarks available: IS, FT, EP, CG, MG, LU, SP and BT, class sizes to choose from: B, C and D, 24 different combinations of benchmarks and class sizes in total, all tests were run on 2 CPUs and 40 logical processors in parallel;

Implementation: OMP-CUDA: benchmarks available: IS, FT, EP, CG, MG, LU, SP and BT, class sizes to choose from: C, D and E, 24 different combinations of benchmarks and class sizes in total, all tests were run on a single GPU with the same grid sizes each time;

Implementation: MPI-Fortran: benchmarks available: IS, FT, EP, CG, MG and LU, class sizes to choose from: B, C and D, 18 different combinations of benchmarks and class sizes in total, all tests were run on 2 CPUs and 32 logical processors in parallel;

Implementation: Horovod-Python: benchmark available: Xception, number of training epochs tested: 1, 3 and 5, 3 different combinations of model training parameters to choose from, all tests were run in configurations with 1, 2 and 4 GPUs, to check training behavior.

Based on these tests and satisfying the aforementioned requirements, we selected the following configurations for subsequent power/energy investigation for the following test types:

CPU on vinnana : ep.D.x – Embarassingly Parallel, class size 'D', lu.C.x – Lower-Upper Gauss-Seidel solver, class size 'C', is.D.x – Integer Sort, class size 'D';

CPU on sanna : bt.C – Block Tri-diagonal solver, class size 'C', is.D – Integer Sort, class size 'D', lu.C – Lower-Upper Gauss-Seidel solver, class size 'C';

GPUs on vinnana : number of epochs for the training of the Xception model was selected to be 1 as the test accuracy of model is relatively high (86.8%) and the execution time is satisfactory (48.5s using 4 GPUs).

GPUs on sanna : lu.D – Lower-Upper Gauss-Seidel solver, class size 'D', sp.D – Scalar Penta-diagonal solver, class size 'D', ep.D – Embarassingly Parallel, class size 'D';

Then mixed versions were selected as follows:

mixed on vinnana : ep.D.x + Xception, is.D.x + Xception, lu.C.x + Xception;
mixed on sanna : bt.C + lu.D, is.D + sp.D, lu.C + ep.D.

We selected the aforementioned benchmarks in order to reflect the variety of real world workload scalability characteristics.

6.3. Results

All the following results are averages from 10 runs. We have also recorded standard deviations, which are reported in tables, for clarity. Before we proceed with presentation of results of the selected configuration runs, we demonstrate values of the measured components, for selected runs on vinnana. Specifically, Table 1 presents power values for vinnana, for Xception run on 1, 2 and 4 GPUs, measured using Yokogawa as well as RAPL and NVML, for CPUs and GPUs, along with execution times, the latter showing scalability of the simulation. Standard deviation values are provided as well showing very good stability of the results.

Table 1. Execution times and power draws; server: **vinnana**, device: **GPUs**, implementation: **Horovod-Python**, benchmark: **Xception**

	GPUs		
Results from 10 runs	1 GPU	2 GPUs	4 GPUs
Avg. Exec. time [s]	133.363	80.633	56.574
Std. dev. of time [s]	0.403	0.360	0.350
(Yokogawa) Avg. power draw [W]	511.645	665.957	905.241
(Yokogawa) Std. dev. of avg. power draw [W]	1.120	2.813	2.909
(CPUs) Avg. power draw [W]	55.354	67.363	81.370
(CPUs) Std. dev. of avg. power draw [W]	0.074	0.203	0.283
(GPU: 0) Avg. power draw [W]	169.563	156.733	135.377
(GPU: 0) Std. dev. of avg. power draw [W]	0.878	0.832	1.089
(GPU: 1) Avg. power draw [W]	14.416	159.318	138.517
(GPU: 1) Std. dev. of avg. power draw [W]	0.097	1.436	1.123
(GPU: 2) Avg. power draw [W]	11.060	10.967	133.840
(GPU: 2) Std. dev. of avg. power draw [W]	0.054	0.120	1.084
(GPU: 3) Avg. power draw [W]	14.637	14.873	137.662
(GPU: 3) Std. dev. of avg. power draw [W]	0.046	0.091	1.053

Similarly, Table 2 presents values of analogous variables for a mixed run of ep.D.x+ Xception using a respective number of 1, 2 and 4 GPUs and 8, 16 and 32 processes for computations. Table 3 presents idle power values for vinnana, measured using Yokogawa for the whole node, using RAPL for CPUs, NVML for GPUs as well as presents the difference between the Yokogawa value and sum of the two latter.

We now proceed with presentation of the measurements for the applications and configurations derived in Section 6.2. Each of the following figures presents values for one computing device type (CPUs, GPUs, mixed CPUs+GPUs) for one system: vinnana or sanna, for a total of 6 figures: vinnana and applications running on CPUs – Figure 1, vinnana and applications running on GPUs and on CPUs+GPUs – Figure 2, sanna and applications running on 1 CPU – Figure 3, sanna and applications running on 2 CPUs – Figure 4, sanna and applications running on GPUs – Figure 5, sanna and applications running on CPUs+GPUs – Figure 6.

Each figure, for each application presents three graphs: average power measured for the whole node by Yokogawa, sum of average powers obtained from RAPL and NVML (in case of sanna measurements include the DRAM component) as well as the offset computed using Equation 1.

Table 2. Execution times and power draws; server: **vinnana**, device: **Hybrid**, implementation: **MPI-Fortran+Horovod-Python**, benchmark: **ep.D.x+Xception**

	Hybrid		
Results from 10 runs (GPU(s) + Processes)	1 + 8	2 + 16	4 + 32
Avg. Exec. time [s]	131.454	80.068	73.330
Std. dev. of time [s]	0.552	1.590	0.355
(Yokogawa) Avg. power draw [W]	561.049	741.187	915.957
(Yokogawa) Std. dev. of avg. power draw [W]	1.145	6.528	2.154
(CPUs) Avg. power draw [W]	99.713	135.506	163.343
(CPUs) Std. dev. of avg. power draw [W]	0.123	0.217	0.221
(GPU: 0) Avg. power draw [W]	171.163	156.899	112.283
(GPU: 0) Std. dev. of avg. power draw [W]	0.676	2.686	0.845
(GPU: 1) Avg. power draw [W]	14.324	159.284	115.304
(GPU: 1) Std. dev. of avg. power draw [W]	0.087	2.510	0.371
(GPU: 2) Avg. power draw [W]	11.058	11.096	111.088
(GPU: 2) Std. dev. of avg. power draw [W]	0.052	0.068	0.945
(GPU: 3) Avg. power draw [W]	14.670	14.960	115.475
(GPU: 3) Std. dev. of avg. power draw [W]	0.034	0.064	1.035

Table 3. Power draw measurements of idle **vinnana** server

	Idle power draw
Measured components	Power draw readings [W]
Sum of entire node idle power draw (Yokogawa)	314.693
Sum of CPUs idle power draw (Linux Perf / Intel RAPL)	46.967
Sum of GPUs idle power draw (NVML)	50.255
Difference between idle node power draw and sum of CPUs and GPUs power draw	217.471

The factory hardware configuration of both sanna and vinnana includes 4 power supplies for each server, for required redundancy. In order to find out the power cost of redundancy we removed 2 power supplies from each server and rerun our test suite for CPU, GPU and mixed CPU+GPU workloads. We observed an average offset of approx. 40 Watts in the overall power consumption between the two configurations, as reported by the Yokogawa power meter.

6.4. Discussion

From all the charts, looking at average power values gathered from application runs, for vinnana, for a given number of threads and/or number of GPUs we can see slightly different results for various workloads, with the exception of very similar power values for ep.D.x+Xception and lu.C.x+Xception for 1 and 2 GPUs (for 4 GPUs values are visibly different), as shown in Figure 2. For sanna, very similar power values can be observed for GPU tests using sp.D and lu.D (Figure 5) and mixed bt.C+lu.D and is.D+sp.D (Figure 6). Some differences can also be observed in the speed of growth of the power (angle) for

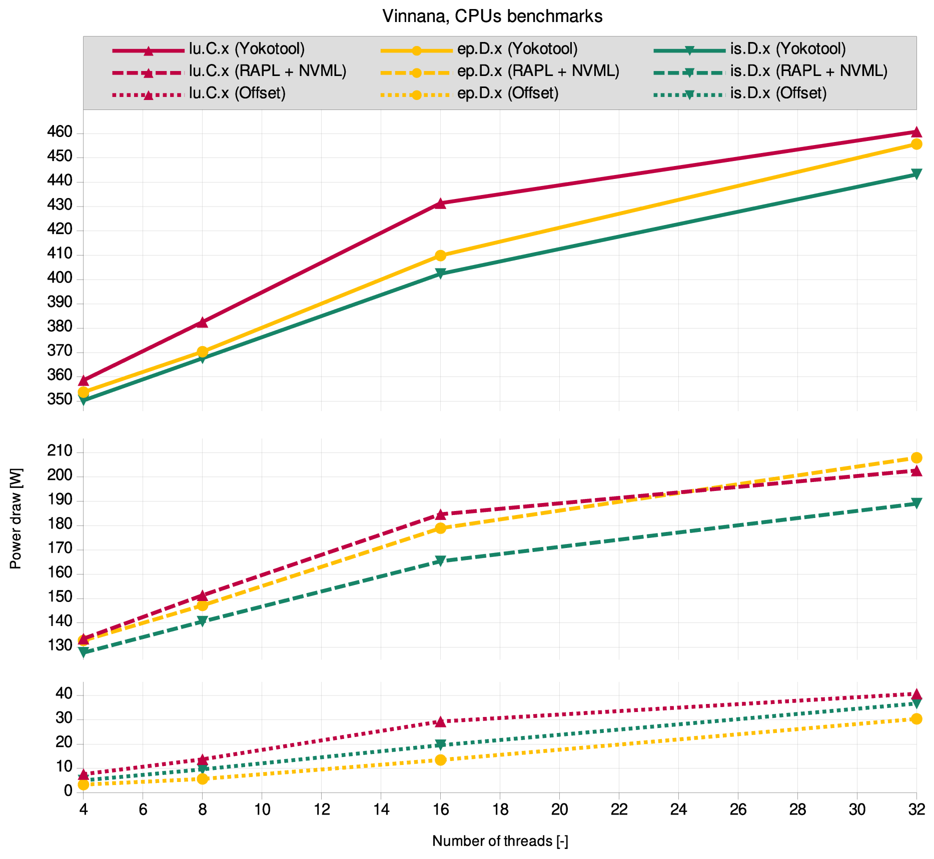


Fig. 1. Measurements using CPU benchmarks on vinnana

various applications, for instance between lu.C.x and is.D.x for vinnana (Figure 1). Interestingly, mixed is.D.x+Xception is not able to put much larger load onto 4 versus 2 GPUs and results from Yokogawa and RAPL+NVML are very consistent here.

The most important observations, in the context of the research goal of this paper, is the observation of the offsets between the hardware (whole node) power measurements from Yokogawa and the sum of measurements from the software APIs i.e. RAPL and NVML. The offsets, considering also differences in idle power values measured using Equation 1, are presented in all Figures 1-6. We can conclude that for all applications running on the CPUs, the offset is growing with an increasing number of threads but is relatively small and reaches only up to approx. 30W for a single CPU using 20 threads (Figure 3), up to approx. 40W using 32 threads on 2 CPUs on vinnana (Figure 1) and 40 threads on sanna (Figure 4). These constitute roughly up to 8.7% of the total power (Yokogawa) for largest numbers of threads in these tests. Those differences include other system components such as fans, disks and potential inaccuracies of RAPL and NVML as the measurements from Yokogawa are considered ground truth. We could also observe

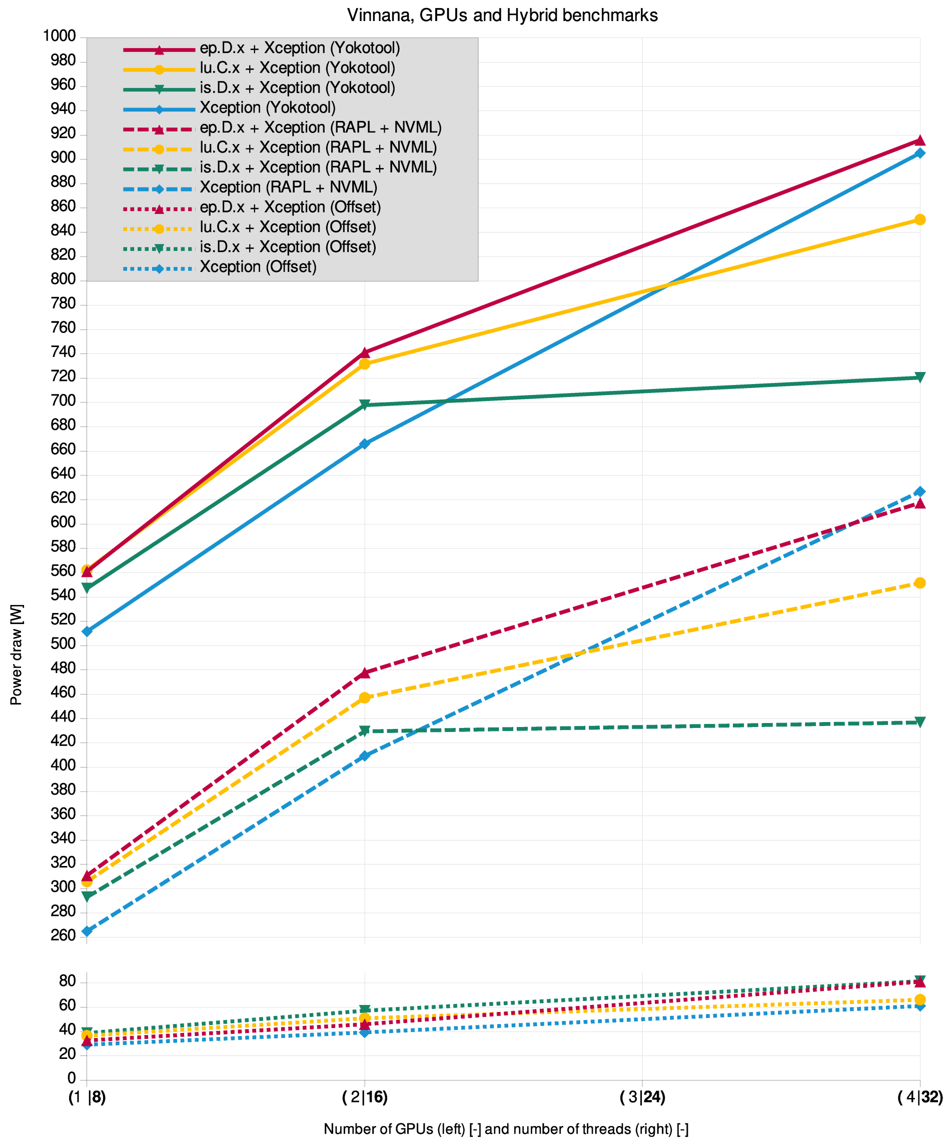


Fig. 2. Measurements using GPU and mixed CPU+GPU benchmarks on vinnana, number of threads and GPUs used is marked on the X axis

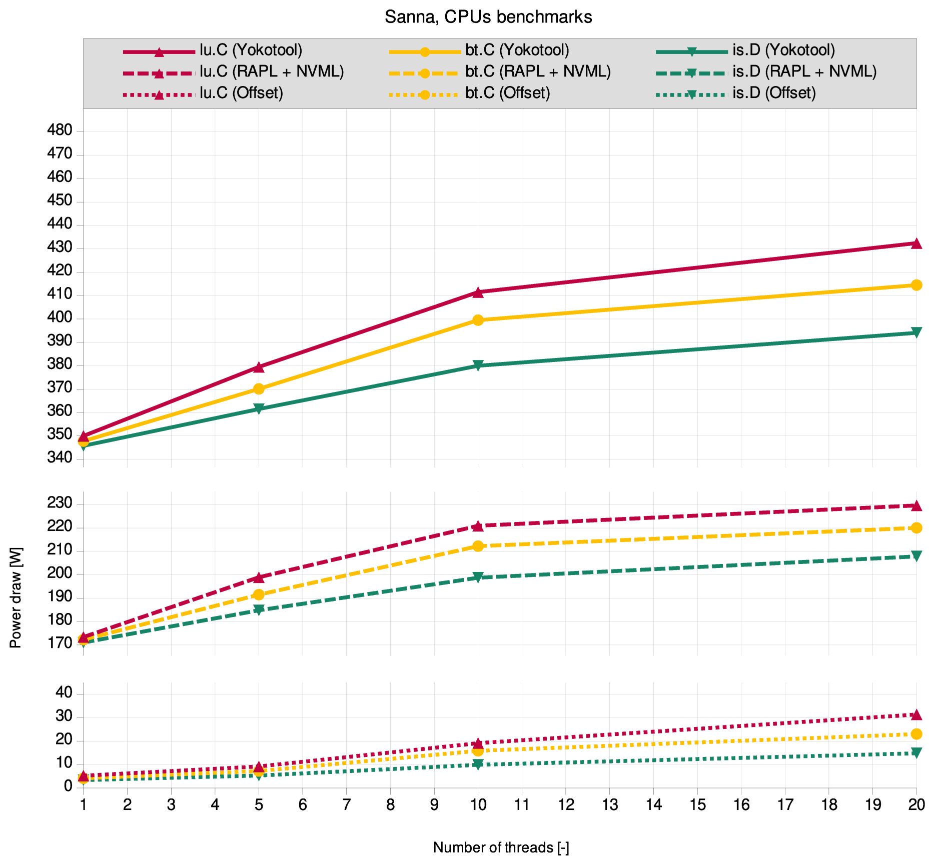


Fig. 3. Measurements using CPU benchmarks on 1 multi-core CPU on sanna

that the growth of the offsets is linear up to the number of available physical cores in the system (20) and then slightly dropping when Hyperthreading is engaged. The similarity of observed patterns and values for offsets on the two similar systems allows us to conclude that the measurements are valid.

For GPU workload (Xception) as well as mixed workloads (also including Xception on the GPU(s)) on vinnana, shown in Figure 2, we see linear growth of the offsets up to approx. 80W reaching the largest percentage of the total power (from Yokogawa) of approx. 11%. We can see that the offsets for all the applications are very similar.

For GPU based workloads as well as CPU+GPU mixed workloads run on sanna, we see a different behaviour. Firstly, we notice that the workloads run on sanna put much more load on the same number of GPUs (4) on both systems, which results from different workloads and different GPU models – Quadro RTX 6000 with a higher TDP in sanna compared to Quadro RTX 5000 in vinnana. Additionally, there are 8 GPUs in sanna, in the same system (motherboard and chassis). This evidently shows in the offsets. We first notice that for these workloads, both GPU and mixed shown in Figure 5 and Figure 6

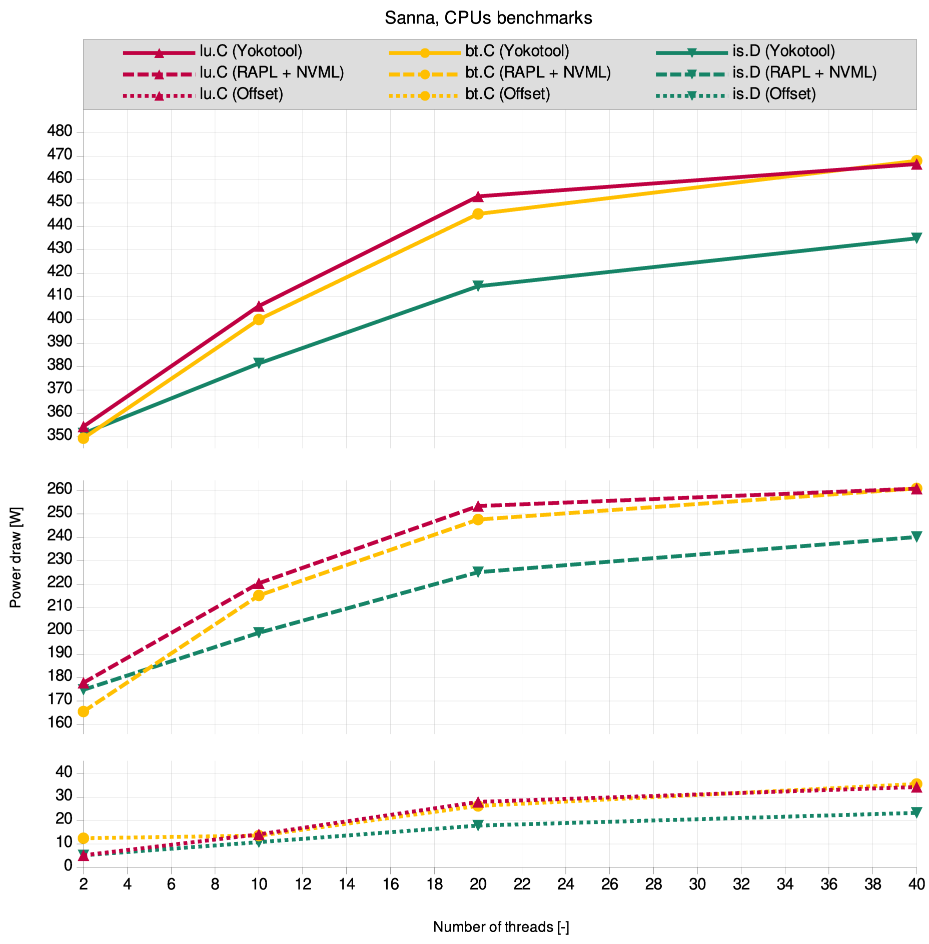


Fig. 4. Measurements using CPU benchmarks on 2 multi-core CPUs on sanna

respectively, RAPL+NVML report average powers growing linearly with an increasing number of GPUs. Additionally, starting with 4 GPUs, both for GPU and mixed workloads, for all applications we see a large consistent jump in the offsets. These then either stay at this level or increase only very slightly for 8 GPUs. These increases are clearly visible in the Yokogawa readings. The values of the offsets for the largest 8 GPU cases range up to approx. 19% of the total power consumed by the node, and up to 22-26% for 4 GPUs, as reported by Yokogawa. These ratios are much lower for 2 GPUs, e.g. approx. 6% for mixed configurations on sanna. We shall note that while performing tests using the Xception application for paper [25], we also compared Yokogawa measurements versus the sum of Intel RAPL and Nvidia NVML and observed the same pattern of power difference increase for 4-8 GPUs. This suggests that the issue is hardware rather than software related.

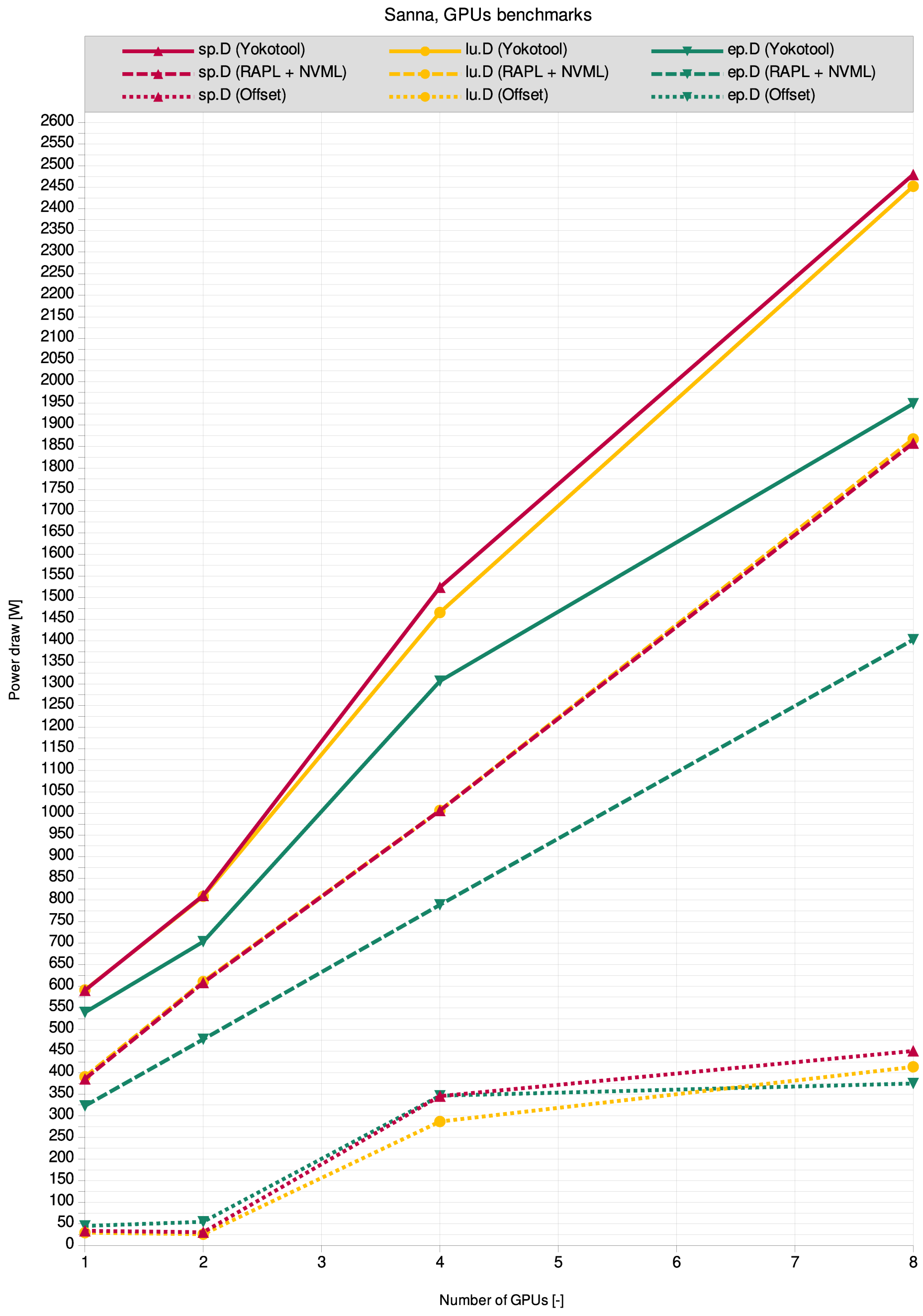


Fig. 5. Measurements using GPU benchmarks on sanna

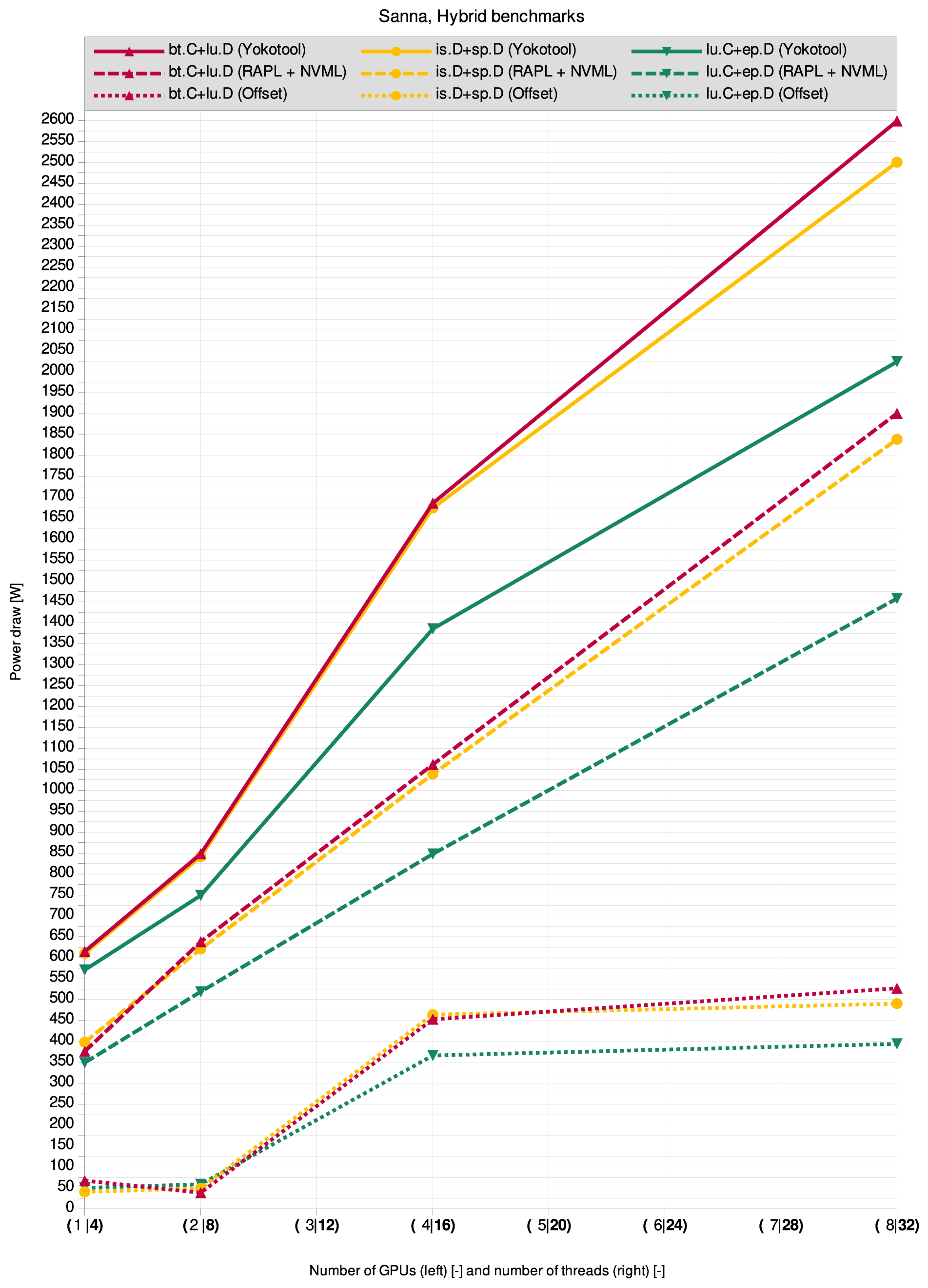


Fig. 6. Measurements using mixed CPU+GPU benchmarks on sanna, number of threads and GPUs used is marked on the X axis

We have compared our results with analysis of the requirements of cooling in high performance computer systems from the literature. Firstly, we note that in the sanna and vinnana systems, there are two rows of carriers that are four columns across, each carrier with two fans, for a total of 16 large chassis fans [21].

Itoh et al. [18] performed detailed analysis of power consumption of fans in an HP blade system. They modeled the speed of fan versus the temperature as well as power consumed versus fan speed (rpm) presenting formula $power = 22.1(rpm/10000)^3 + 8.2$. They concluded that, in their case, fans consumed approx 0.8$\sim$1kW which amounted to 16$\sim$20% of the system. Similarly, Kennedy [22] analyzed the 1U and 2U system fan power consumption as a percentage of total power consumption, across 9 workloads. The 1U fan power consumption was approx. 1% higher and amounted to 15$\sim$16% of the total power. Neudorfer [31] stated that server fans can consume 10% to 15% or more of the total power drawn by the server. Based on those findings, taking into account the our offsets include also other system components such as disks, we believe these are in line with the aforementioned observations.

7. Summary and Future Work

In the paper, we performed detailed comparison of hardware and software based power/ energy measurement methods using the hardware power meter Yokogawa WT-310E as well as Intel RAPL and Nvidia NVML interfaces respectively. We performed tests using 2 systems, one with 2 Intel Xeon CPUs and 8 Nvidia Quadro RTX 6000 GPUs and the second 2 Intel Xeon CPUs and 4 Nvidia Quadro RTX 5000 GPUs. For thorough comparison we used selected kernels from NAS Parallel Benchmarks for CPUs and GPUs and Xception deep neural network training using several GPUs. Tests were conducted for CPU, GPU as well as mixed CPUs+GPUs configurations, using 1-40 threads for the CPUs and 1-8 GPUs.

We shall note that the software power measurements from Intel RAPL and Nvidia NVML only capture the CPU, GPU and DRAM components without e.g. cooling and disks. We have determined that the offset, computed using formula 1 in Section 4, grows with an increasing load (numbers of CPU threads and GPUs used) and amounts to roughly 8.7% of the total system power for CPU workloads. For GPU and mixed workloads on the system with 4 GPUs the offset reaches up to 11% of the total system power. At the extreme end, under the highest 2 CPUs + 4-8 GPUs load on the system with 8 Quadro RTX 6000, it rises up to roughly 19-26% of the total system power. In the high-load range (multi-GPU workloads on sanna) obtained offset patterns clearly indicate node cooling system saturation point.

In the future, we plan to extend the scope of tests in terms of applications and systems. Additionally, we plan to perform a similar comparison with the power measurements obtained from the system platform as well, when provided. We also plan to correlate values of system metrics corresponding to the loads of CPUs and GPUs with power measurement offsets. That would allow runtime estimation of whole node power draw using previously obtained node characteristics.

A. Detailed Results from sanna

For clarity, this appendix section contains detailed results of measurements of selected benchmarks, including execution times, power measured for individual computing devices and the whole sanna node, as well as standard deviation values. We chose to include details for sanna rather than vinnana as sanna is better equipped and reveals cooling limitations in a more pronounced way. Table 4 contains measurements for the lu.C benchmark executed in parallel using various numbers of threads on a single multi-core CPU, Table 5 presents analogus results using 2 multi-core CPUs, Table 6 details measurements from the ep.D benchmarks using GPUs and finally Table 7 contains values corresponding to mixed runs of lu.C and ep.D on both CPUs and GPUs.

Table 4. Execution times and power draws; server: **sanna**, device: **1 CPU**, implementation: **OMP-CPP**, benchmark: **lu.C**

	1 CPU			
Results from 10 runs	1 Thread	5 Threads	10 Threads	20 Threads
Avg. Exec. time [s]	709.838	146.984	76.672	64.195
Std. dev. of time [s]	1.198	0.124	0.201	0.196
(Yokogawa) Avg. power draw [W]	349.991	379.51	411.501	432.449
(Yokogawa) Std. dev. of avg. power draw [W]	0.085	0.212	0.829	1.553
(CPU: 0) Avg. power draw [W]	34.419	54.615	74.217	82.381
(CPU: 0) Std. dev. of avg. power draw [W]	0.221	0.053	0.079	0.055
(CPU: 1) Avg. power draw [W]	28.919	29.018	28.887	28.803
(CPU: 1) Std. dev. of avg. power draw [W]	0.297	0.075	0.042	0.030
(RAM) Avg. power draw [W]	23.574	27.971	30.029	31.413
(RAM) Std. dev. of avg. power draw [W]	0.093	0.030	0.053	0.046
(GPU: 0) Avg. power draw [W]	5.384	5.414	5.378	5.387
(GPU: 0) Std. dev. of avg. power draw [W]	0.077	0.101	0.073	0.056
(GPU: 1) Avg. power draw [W]	9.691	10.083	10.046	9.759
(GPU: 1) Std. dev. of avg. power draw [W]	0.180	0.139	0.185	0.186
(GPU: 2) Avg. power draw [W]	13.812	13.757	13.811	13.945
(GPU: 2) Std. dev. of avg. power draw [W]	0.164	0.148	0.041	0.190
(GPU: 3) Avg. power draw [W]	17.234	17.248	17.240	17.233
(GPU: 3) Std. dev. of avg. power draw [W]	0.014	0.017	0.012	0.011
(GPU: 4) Avg. power draw [W]	4.045	4.037	4.017	4.025
(GPU: 4) Std. dev. of avg. power draw [W]	0.011	0.017	0.006	0.011
(GPU: 5) Avg. power draw [W]	18.450	18.661	18.709	18.585
(GPU: 5) Std. dev. of avg. power draw [W]	0.111	0.136	0.164	0.117
(GPU: 6) Avg. power draw [W]	6.469	6.393	6.593	6.621
(GPU: 6) Std. dev. of avg. power draw [W]	0.077	0.082	0.108	0.079
(GPU: 7) Avg. power draw [W]	11.371	11.745	12.035	11.495
(GPU: 7) Std. dev. of avg. power draw [W]	0.280	0.249	0.191	0.147

Table 5. Execution times and power draws; server: **sanna**, device: **2 CPUs**, implementation: **OMP-CPP**, benchmark: **lu.C**

	1 CPU			
Results from 10 runs	2 Threads	10 Threads	20 Threads	40 Threads
Avg. Exec. time [s]	518.970	75.024	46.207	44.716
Std. dev. of time [s]	26.472	0.910	0.385	1.309
(Yokogawa) Avg. power draw [W]	354.402	405.909	452.840	466.669
(Yokogawa) Std. dev. of avg. power draw [W]	0.550	0.808	1.639	4.044
(CPU: 0) Avg. power draw [W]	33.168	52.455	67.115	70.499
(CPU: 0) Std. dev. of avg. power draw [W]	0.155	0.204	0.163	0.524
(CPU: 1) Avg. power draw [W]	33.589	51.560	67.221	70.781
(CPU: 1) Std. dev. of avg. power draw [W]	0.307	0.247	0.215	0.605
(RAM) Avg. power draw [W]	23.927	28.316	30.873	31.369
(RAM) Std. dev. of avg. power draw [W]	0.096	0.132	0.117	0.234
(GPU: 0) Avg. power draw [W]	5.559	5.514	5.508	5.538
(GPU: 0) Std. dev. of avg. power draw [W]	0.204	0.114	0.093	0.061
(GPU: 1) Avg. power draw [W]	10.276	10.210	10.480	10.347
(GPU: 1) Std. dev. of avg. power draw [W]	0.298	0.185	0.174	0.064
(GPU: 2) Avg. power draw [W]	13.689	13.966	13.833	13.863
(GPU: 2) Std. dev. of avg. power draw [W]	0.125	0.126	0.166	0.070
(GPU: 3) Avg. power draw [W]	17.229	17.238	17.226	17.256
(GPU: 3) Std. dev. of avg. power draw [W]	0.034	0.017	0.014	0.011
(GPU: 4) Avg. power draw [W]	4.004	4.012	4.039	4.061
(GPU: 4) Std. dev. of avg. power draw [W]	0.043	0.016	0.013	0.023
(GPU: 5) Avg. power draw [W]	18.408	18.451	18.554	18.537
(GPU: 5) Std. dev. of avg. power draw [W]	0.183	0.121	0.097	0.097
(GPU: 6) Avg. power draw [W]	6.278	6.446	6.314	6.491
(GPU: 6) Std. dev. of avg. power draw [W]	0.381	0.087	0.113	0.097
(GPU: 7) Avg. power draw [W]	11.684	12.216	12.250	12.131
(GPU: 7) Std. dev. of avg. power draw [W]	0.367	0.140	0.144	0.212

Table 6. Execution times and power draws; server: **sanna**, device: **GPUs**, implementation: **OMP-CUDA**, benchmark: **ep.D**

	1 CPU			
Results from 10 runs	1 GPU	2 GPUs	4 GPUs	8 GPUs
Avg. Exec. time [s]	28.056	28.212	28.350	28.975
Std. dev. of time [s]	0.080	0.031	0.051	0.133
(Yokogawa) Avg. power draw [W]	539.594	703.358	1306.636	1949.073
(Yokogawa) Std. dev. of avg. power draw [W]	8.704	6.079	15.716	11.492
(CPU: 0) Avg. power draw [W]	33.286	37.151	37.555	45.274
(CPU: 0) Std. dev. of avg. power draw [W]	0.065	0.639	0.223	0.092
(CPU: 1) Avg. power draw [W]	28.970	28.776	37.179	44.509
(CPU: 1) Std. dev. of avg. power draw [W]	0.152	0.231	0.095	0.073
(RAM) Avg. power draw [W]	21.765	21.899	22.030	21.994
(RAM) Std. dev. of avg. power draw [W]	0.077	0.094	0.106	0.074
(GPU: 0) Avg. power draw [W]	156.550	156.842	153.124	150.186
(GPU: 0) Std. dev. of avg. power draw [W]	1.426	0.428	0.299	0.892
(GPU: 1) Avg. power draw [W]	10.206	160.103	158.563	155.258
(GPU: 1) Std. dev. of avg. power draw [W]	0.151	1.263	0.448	0.692
(GPU: 2) Avg. power draw [W]	13.757	13.844	161.103	158.964
(GPU: 2) Std. dev. of avg. power draw [W]	0.138	0.106	1.131	1.452
(GPU: 3) Avg. power draw [W]	17.208	17.205	174.830	172.491
(GPU: 3) Std. dev. of avg. power draw [W]	0.019	0.023	1.426	0.342
(GPU: 4) Avg. power draw [W]	4.055	4.047	4.107	153.803
(GPU: 4) Std. dev. of avg. power draw [W]	0.013	0.011	0.012	1.957
(GPU: 5) Avg. power draw [W]	18.628	18.739	19.708	168.175
(GPU: 5) Std. dev. of avg. power draw [W]	0.145	0.152	0.135	0.994
(GPU: 6) Avg. power draw [W]	5.995	5.979	6.795	158.945
(GPU: 6) Std. dev. of avg. power draw [W]	0.081	0.114	0.152	1.391
(GPU: 7) Avg. power draw [W]	12.572	12.626	13.555	172.951
(GPU: 7) Std. dev. of avg. power draw [W]	0.215	0.201	0.163	1.657

Table 7. Execution times and power draws; server: **sanna**, device: **Hybrid**, implementation: **OMP-CPP+OMP-CUDA**, benchmark: **lu.C+ep.D**

	1 CPU			
Results from 10 runs	1 + 4	2 + 8	4 + 16	8 + 32
Avg. Exec. time [s]	27.974	28.101	28.314	29.211
Std. dev. of time [s]	0.082	0.072	0.076	0.186
(Yokogawa) Avg. power draw [W]	570.861	748.939	1385.889	2024.047
(Yokogawa) Std. dev. of avg. power draw [W]	8.010	9.171	26.194	11.160
(CPU: 0) Avg. power draw [W]	53.715	72.342	65.580	71.738
(CPU: 0) Std. dev. of avg. power draw [W]	0.106	5.246	5.428	0.738
(CPU: 1) Avg. power draw [W]	29.490	29.483	65.091	72.040
(CPU: 1) Std. dev. of avg. power draw [W]	0.108	0.069	7.089	0.824
(RAM) Avg. power draw [W]	27.094	29.173	27.993	29.224
(RAM) Std. dev. of avg. power draw [W]	0.086	2.041	2.760	0.385
(GPU: 0) Avg. power draw [W]	157.522	157.090	153.753	150.166
(GPU: 0) Std. dev. of avg. power draw [W]	1.623	1.619	1.082	0.598
(GPU: 1) Avg. power draw [W]	10.251	159.154	157.782	154.078
(GPU: 1) Std. dev. of avg. power draw [W]	0.181	0.775	1.027	0.878
(GPU: 2) Avg. power draw [W]	14.145	14.103	161.310	158.264
(GPU: 2) Std. dev. of avg. power draw [W]	0.191	0.182	1.294	0.991
(GPU: 3) Avg. power draw [W]	17.251	17.239	173.964	171.584
(GPU: 3) Std. dev. of avg. power draw [W]	0.026	0.031	1.835	1.215
(GPU: 4) Avg. power draw [W]	3.966	3.973	3.996	154.742
(GPU: 4) Std. dev. of avg. power draw [W]	0.005	0.013	0.012	0.731
(GPU: 5) Avg. power draw [W]	18.808	18.940	19.680	167.668
(GPU: 5) Std. dev. of avg. power draw [W]	0.164	0.194	0.344	0.620
(GPU: 6) Avg. power draw [W]	6.049	6.233	6.476	156.762
(GPU: 6) Std. dev. of avg. power draw [W]	0.081	0.346	0.107	0.808
(GPU: 7) Avg. power draw [W]	10.969	11.067	12.323	171.879
(GPU: 7) Std. dev. of avg. power draw [W]	0.157	0.220	0.261	1.756

Acknowledgments. This work is partially supported by CERCIRAS COST Action CA19135 funded by COST. The research was supported in part by project "Cloud Artificial Intelligence Service Engineering (CAISE) platform to create universal and smart services for various application areas", No. KPOD.05.10-IW.10-0005/24, as part of the European IPCEI-CIS program, financed by NRRP (National Recovery and Resilience Plan) funds.

References

1. Alqurashi, F.S., Al-Hashimi, M.: An experimental approach to estimation of the energy cost of dynamic branch prediction in an intel high-performance processor. Computers 12(7) (2023), https://www.mdpi.com/2073-431X/12/7/139
2. Arafa, Y., ElWazir, A., Elkanishy, A., Aly, Y., Elsayed, A., Badawy, A.H., Chennupati, G., Eidenbenz, S., Santhi, N.: Nvidia gpgpus instructions energy consumption. In: 2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS). pp. 110–112 (2020)
3. Araujo, G., Griebler, D., Rockenbach, D.A., Danelutto, M., Fernandes, L.G.: Nas parallel benchmarks with cuda and beyond. Software: Practice and Experience 53(1), 53–80 (2023), https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.3056
4. Araujo, G.A.d., Griebler, D., Danelutto, M., Fernandes, L.G.: Efficient nas parallel benchmark kernels with cuda. In: 2020 28th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP). pp. 9–16 (2020)
5. Aslan, B., Yilmazer-Metin, A.: A study on power and energy measurement of nvidia jetson embedded gpus using built-in sensor. In: 2022 7th International Conference on Computer Science and Engineering (UBMK). pp. 1–6 (2022)
6. Bailey, D.H.: NAS Parallel Benchmarks, pp. 1254–1259. Springer US, Boston, MA (2011), https://doi.org/10.1007/978-0-387-09766-4_133
7. Bityutskiy, A., Laakso, A., Correia, H.: Yokotool, https://github.com/intel/yoko-tool, accessed on March 5th 2024
8. Burtscher, M., Zecena, I., Zong, Z.: Measuring gpu power with the k20 built-in sensor. In: Proceedings of Workshop on General Purpose Processing Using GPUs. p. 28–36. GPGPU-7, Association for Computing Machinery, New York, NY, USA (2014), https://doi.org/10.1145/2588768.2576783
9. Czarnul, P.: Parallel Programming for Modern High Performance Computing Systems. CRC Press, Taylor & Francis (2018), iSBN 9781138305953
10. Czarnul, P., Proficz, J., Krzywaniak, A.: Energy-aware high-performance computing: Survey of state-of-the-art tools, techniques, and environments. Sci. Program. 2019, 8348791:1–8348791:19 (2019), https://doi.org/10.1155/2019/8348791
11. Desrochers, S., Paradis, C., Weaver, V.M.: A validation of dram rapl power measurements. In: Proceedings of the Second International Symposium on Memory Systems. p. 455–470. MEMSYS '16, Association for Computing Machinery, New York, NY, USA (2016), https://doi.org/10.1145/2989081.2989088
12. Fahad, M., Shahid, A., Manumachu, R.R., Lastovetsky, A.: A comparative study of methods for measurement of energy of computing. Energies 12(11) (2019), https://www.mdpi.com/1996-1073/12/11/2204
13. Ferro, M., Yokoyama, A., Klõh, V., Silva, G., Gandra, R., Bragança, R., Bulcão, A., Schulze, B.: Analysis of gpu power consumption using internal sensors. In: Anais do XVI Workshop em Desempenho de Sistemas Computacionais e de Comunicação. SBC, Porto Alegre, RS, Brasil (2017), https://sol.sbc.org.br/index.php/wperformance/article/view/3360

14. Hähnel, M., Döbel, B., Völp, M., Härtig, H.: Measuring energy consumption for short code paths using rapl. SIGMETRICS Perform. Eval. Rev. 40(3), 13–17 (jan 2012), https://doi.org/10.1145/2425248.2425252
15. Ikram, M.J., Abulnaja, O.A., Saleh, M.E., Al-Hashimi, M.A.: Measuring power and energy consumption of programs running on kepler gpus. In: 2017 Intl Conf on Advanced Control Circuits Systems (ACCS) Systems & 2017 Intl Conf on New Paradigms in Electronics & Information Technology (PEIT). pp. 18–25 (2017)
16. Ilsche, T.: Energy Measurements of High Performance Computing Systems: From Instrumentation to Analysis. Ph.D. thesis, Technischen Universität Dresden (March 2020), https://tud.qucosa.de/api/qucosa%3A71600/attachment/ATT-0/
17. Intel Corporation: Running average power limit energy reporting / cve-2020-8694 , cve-2020-8695 / intel-sa-00389 (February 2022), https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/advisory-guidance/running-average-power-limit-energy-reporting.html
18. Itoh, S., Kodama, Y., Shimizu, T., Sekiguchi, S., Nakamura, H., Mori, N.: Power consumption and efficiency of cooling in a data center. In: 2010 11th IEEE/ACM International Conference on Grid Computing. pp. 305–312 (2010)
19. Jay, M., Ostapenco, V., Lefevre, L., Trystram, D., Orgerie, A.C., Fichel, B.: An experimental comparison of software-based power meters: focus on cpu and gpu. In: 2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing (CCGrid). pp. 106–118 (2023)
20. Kavanagh, R., Djemame, K.: Rapid and accurate energy models through calibration with ipmi and rapl. Concurrency and Computation: Practice and Experience 31(13), e5124 (2019), https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.5124, e5124 cpe.5124
21. Kennedy, P.: Inspur systems nf5468m5 review 4u 8x gpu server (March 2019), serveTheHome, https://www.servethehome.com/inspur-systems-nf5468m5-review-4u-8x-gpu-server/
22. Kennedy, P.: Deep dive into lowering server power consumption (February 2022), serveTheHome, https://www.servethehome.com/deep-dive-into-lowering-server-power-consumption-intel-inspur-hpe-dell-emc/
23. Khan, K.N., Hirki, M., Niemi, T., Nurminen, J.K., Ou, Z.: Rapl in action: Experiences in using rapl for power measurements. ACM Trans. Model. Perform. Eval. Comput. Syst. 3(2) (mar 2018), https://doi.org/10.1145/3177754
24. Kocot, B., Czarnul, P., Proficz, J.: Energy-aware scheduling for high-performance computing systems: A survey. Energies 16(2) (2023), https://www.mdpi.com/1996-1073/16/2/890
25. Koszczał, G., Dobrosolski, J., Matuszek, M., Czarnul, P.: Performance and energy aware training of a deep neural network in a multi-gpu environment with power capping. In: Zeinalipour, D., Blanco Heras, D., Pallis, G., Herodotou, H., Trihinas, D., Balouek, D., Diehl, P., Cojean, T., Fürlinger, K., Kirkeby, M.H., Nardellli, M., Di Sanzo, P. (eds.) Euro-Par 2023: Parallel Processing Workshops. pp. 5–16. Springer Nature Switzerland, Cham (2024)
26. Krzywaniak, A., Czarnul, P., Proficz, J.: Depo: A dynamic energy-performance optimizer tool for automatic power capping for energy efficient high-performance computing. Software: Practice and Experience 52(12), 2598–2634 (2022), https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.3139
27. Krzywaniak, A., Czarnul, P., Proficz, J.: Dynamic gpu power capping with online performance tracing for energy efficient gpu computing using depo tool. Future Generation Computer Systems 145, 396–414 (2023), https://www.sciencedirect.com/science/article/pii/S0167739X23001267

28. Lang, J., Rünger, G.: High-resolution power profiling of gpu functions using low-resolution measurement. In: Wolf, F., Mohr, B., an Mey, D. (eds.) Euro-Par 2013 Parallel Processing. pp. 801–812. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
29. Lucas, J., Juurlink, B.: Alupower: Data dependent power consumption in gpus. In: 2016 IEEE 24th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS). pp. 95–104 (2016)
30. Löff, J., Griebler, D., Mencagli, G., Araujo, G., Torquati, M., Danelutto, M., Fernandes, L.G.: The nas parallel benchmarks for evaluating c++ parallel programming frameworks on shared-memory architectures. Future Generation Computer Systems 125, 743–757 (2021), https://www.sciencedirect.com/science/article/pii/S0167739X21002831
31. Neudorfer, J.: Optimizing server energy efficiency (February 2009), techTarget, https://www.techtarget.com/searchdatacenter/tip/Optimizing-server-energy-efficiency
32. NVIDIA Corporation: Nvidia management library (nvml), https://developer.nvidia.com/management-library-nvml, accessed Sep 2024
33. Paniego, J.M., Gallo, S., Pi Puig, M., Chichizola, F., De Giusti, L., Balladini, J.: Analysis of rapl energy prediction accuracy in a matrix multiplication application on shared memory. In: De Giusti, A.E. (ed.) Computer Science – CACIC 2017. pp. 37–46. Springer International Publishing, Cham (2018)
34. Raffin, G., Trystram, D.: Dissecting the software-based measurement of cpu energy consumption: a comparative analysis (2024)
35. Sen, S., Imam, N., Hsu, C.H.: Quality assessment of gpu power profiling mechanisms. In: 2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). pp. 702–711 (2018)
36. Shahid, A., Fahad, M., Manumachu, R.R., Lastovetsky, A.: Improving the accuracy of energy predictive models for multicore cpus by combining utilization and performance events model variables. Journal of Parallel and Distributed Computing 151, 38–51 (2021), https://www.sciencedirect.com/science/article/pii/S0743731521000137
37. Shalavi, N., Khoshsirat, A., Stellini, M., Zanella, A., Rossi, M.: Accurate calibration of power measurements from internal power sensors on nvidia jetson devices. In: 2023 IEEE International Conference on Edge Computing and Communications (EDGE). pp. 166–170 (2023)
38. Tröpgen, H., Bielert, M., Ilsche, T.: Evaluating the energy measurements of the ibm power9 on-chip controller. In: Proceedings of the 2023 ACM/SPEC International Conference on Performance Engineering. p. 67–76. ICPE '23, Association for Computing Machinery, New York, NY, USA (2023), https://doi.org/10.1145/3578244.3583729
39. Vatjus-Anttila, J.M., Koskela, T., Hickey, S.: Power consumption model of a mobile gpu based on rendering complexity. In: 2013 Seventh International Conference on Next Generation Mobile Apps, Services and Technologies. pp. 210–215 (2013)
40. Yang, Z., Adamek, K., Armour, W.: Part-time power measurements: nvidia-smi's lack of attention (2023)
41. Yokogawa: WT310E/WT310EH/WT332E/WT333E Digital Power Meter. User's Manual (October 2017), 2nd edition, IM WT310E-01EN, https://cdn.tmi.yokogawa.com/IMWT310E-01EN.pdf
42. Zhang, H., Hoffmann, H.: A quantitative evaluation of the rapl power control system. In: Proceedings of the 10th International Workshop on Feedback Computing (2015), https://api.semanticscholar.org/CorpusID:13950838

Grzegorz Koszczał is a research assistant at Department of Computer Systems Architecture, Faculty of Electronics, Telecommunications and Informatics, Gdańsk University of Technology. He completed his MSc degree in computer science in 2023 and is currently doing research on performance-energy aspects in HPC and cloud systems.

Mariusz Matuszek is an assistant professor at Department of Computer Systems Architecture, Faculty of Electronics, Telecommunications and Informatics, Gdańsk University of Technology. His main research areas focus on intelligent distributed systems, in connection with power efficient computing. Most of his professional life is devoted to academic research, with a short interruption by an engineering position with Philips early on.

Paweł Czarnul is an associate professor, v-Dean for Cooperation & Promotion and Head of Computer Architecture Department at Faculty of Electronics, Telecommunications and Informatics, Gdańsk University of Technology. His research interests include: high performance computing, distributed systems, and artificial intelligence. He is an author of over 130 publications in the area of parallel and distributed processing, including 2 books. He was a PI or participated in over 20 research, didactic or organizational projects.

Received: July 22, 2024; Accepted: November 10, 2024.

Computer Science and Information Systems 22(2):591–621 https://doi.org/10.2298/CSIS240724024W

Manatee: A Multicore Interference Analysis Tool for Embedded SoC Evaluation[⋆]

Axel Wiedemann[⋆⋆], Florian Haas, and Sebastian Altmeyer

Faculty of Applied Computer Science, University of Augsburg
Universitätsstr. 6a, 86159 Augsburg, Germany
{wiedemann,-,altmeyer}@es-augsburg.de

Abstract. Interferences on shared resources are the main factor limiting the employment of multicore architectures in many embedded use cases. Research on these interferences and enhancements, for example in memory hierarchies, could alleviate this restriction. This however requires more awareness of contention for shared resources during the design and development process of System on Chips (SoCs). As an answer we present the concept of a tool which brings this awareness to the RISC-V hardware development framework Chipyard. It extends Chipyard's agile development focus by adding the capabilities for quick feedback on changes regarding shared resource contention. A partial realisation further allows first tests and evaluation on use case basis.

Keywords: parallel real-time system, memory hierarchy, FPGA prototyping framework.

1. Introduction

The performance of multicore processors is strongly desired in various domains of embedded systems to satisfy the increasing demand for computational power. Complex algorithms and software systems, e. g. in autonomous driving, benefit from high-performance general-purpose shared-memory multicores. However, these processors do not meet the typical requirements on real-time and safety, and thus cannot be used without performance-degrading and laborious software mechanisms. Elaborate methods in such systems have been developed to further improve the average-case performance of the processor, for example the increasing depth of the memory hierarchy. These and the shared resources, like last-level caches, buses, and main memory, result in the ultimate challenge of calculating tight WCET (worst-case execution time) bounds for the tasks in a time-critical system [41].

The crucial problem is the missing guaranteed freedom from interferences between tasks that run on separate cores. Thus, an arbitrary low-priority task is able to influence the timing behaviour of another, potentially high-priority task on a different core. This can happen through accesses on shared resources, for example shared caches or the main memory [26]. As a consequence, a schedulability analysis of the overall system with only minimal overestimation becomes nearly impossible for more than a few cores and deeper memory hierarchies.

[⋆] This is an extended version of the conference paper [20]

[⋆⋆] Corresponding author

The general objective of research on this topic is to facilitate predictable performance, with minimal over-estimation of timing bounds, by reducing the sources of potential interferences on shared resources. Existing software-based approaches, e. g. performance counter monitors [16], or program modification during compilation, are limited, as they can either only detect excessing interferences, or are required to be applied to all tasks of the system. Thus, hardware mechanisms promise a better lever to control the behaviour of any task on the system. However, to research hardware-implemented methods, a proper evaluation platform is required. For example, a hardware implementation of a memory bandwidth reservation mechanism like MemGuard [46] could be evaluated and compared with other approaches. To research potential improvements on shared resource accesses under timing constraints, a realistic model of a typical memory hierarchy is needed in the first place. Microarchitecture simulators with multicore configurations exist, but their processor-centric design does not support for a prototype implementation and a realistic evaluation. Further, the evaluation system needs to be capable of executing realistic benchmarks, for prototyping different ideas, as well as for a thorough evaluation of their impact on the performance.

Previous work [21,19] focused mostly on fault tolerance of parallel systems [30], but the research always involved shared-memory systems. Different systems have been used to evaluate the proposed methods, from software-only approaches on typical desktop and server hardware, over the gem5 simulator [7,25], to FPGA prototypes. As a side effect of the conducted implementations and evaluations, some experience with diverse platforms has been collected. The work on a software-only fault tolerance mechanism [21,19] showed the numerous restrictions of an unmodifiable hardware implementation. To overcome these limitations, later research was undertaken on the gem5 microarchitecture simulator, where a customised hardware transactional memory was built into the memory hierarchy [4,33]. However, since the simulator focuses on the detailed simulation of the processor cores itself, it provides only a rather functional memory hierarchy with limited timing accuracy. Switching to an FPGA prototype with multiple MicroBlaze softcores [3] showed the difficulties of integrating hardware and software parts with non-open processor cores. Overall, these experiences affirm the demand for an open system to prototype and evaluate memory hierarchies for future research ideas.

This paper introduces Manatee, a Multicore interference ANAlysis Tool for Embedded soc Evaluation, aiming to finally serve this need in the context of both simulation and prototyping on an FPGA. Manatee is a tool and framework based on Chipyard [2], which supports design and evaluation of full-system hardware, using the Rocket Chip generator [5] and its in-order RISC-V CPUs. The main benefit of Chipyard is the configurability and customisability of the involved modules. The interconnects can be replaced with a network on a chip (NoC) to research on manycore systems, or a combination of both with shared-memory clusters connected through an NoC.

While Chipyard's hardware evaluation capabilities are limited to provide functional correctness, Manatee allows for evaluations in terms of interference potential on contested resources during workload execution. Manatee also provides the necessary structure and level of automation to perform the required measurements and curates results in insightful and accessible fashion.

Here we provide the extended version of the original proposal of [20]. As a part of this extension we build on the initial concept by providing further specifications and realise its core by implementing significant portions. Further, we provide the above mentioned measurement automation and visualisation capabilities. Additionally, we conduct tests on basic SoC designs in order to assess the framework itself. These tests will also enable an exemplary analysis of protocolled L2 cache accesses.

As a consequence, Manatee enables for the first time ever an agile SoC prototyping workflow which empowers its users with a steady awareness of multicore interferences. This in turn prevents design choices hurting WCETs and helps to discover robust architectures well suited for embedded systems with strict real-time requirements.

In the next section we will explain the basics with Chipyard and its components further. After, related work is introduced to show alternatives to this approach and used components. In the fourth section, the tool's design and key parts of its implementation will be described in detail. Then, in the fifth section, the framework is evaluated before the conclusion closes but also expands with possible future work.

2. Basics

Manatee builds upon existing open-source projects that have been developed in recent years around the prevalent RISC-V architecture.

2.1. Chipyard

Chipyard [2] simplifies the process of designing full-system hardware by integrating all necessary parts from CPU cores to supplementing logic to connect the devices of an FPGA evaluation board. Fig. 1 illustrates the individual parts of Chipyard: Processor cores can be created for example with the Rocket Chip generator [5] (see next section), which generates configurable and customisable cores that implement the RISC-V instruction set [40,39], either in-order Rocket cores, or the more complex and powerful out-of-order BOOM cores. Beside the L1 caches provided by the Rocket Chip generator, secondary level caches and different kinds of interconnecting buses can be generated. There is also code provided to connect to and communicate with peripheral devices like UART and JTAG. The generated Verilog code can be further compiled with Verilator [36] for a simulation of the overall system, or with FireSim, which additionally allows to simulate DDR3 main memory.

2.2. Rocket Chip Generator

The Rocket Chip generator [5] produces designs of an SoC with multiple processor cores, a memory hierarchy, and interconnects. Fig. 2 depicts a generated chip with four processor tiles, consisting of an in-order Rocket RISC-V core[1] and L1 instruction and data caches, L2 cache banks with the memory bus, and additional buses for peripheral devices, DMA devices, and control units like the boot ROM and interrupt controllers. All processor tiles and all individual buses are connected through a shared system bus, which is typically implemented as a crossbar, but can also be configured as a ring bus.

[1] https://chipyard.readthedocs.io/en/stable/Generators/Rocket.html

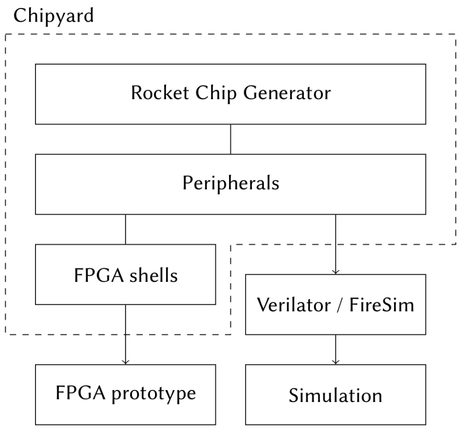


Fig. 1. Overview of generators of Chipyard [2]. The resulting code can be synthesised for an FPGA or simulated

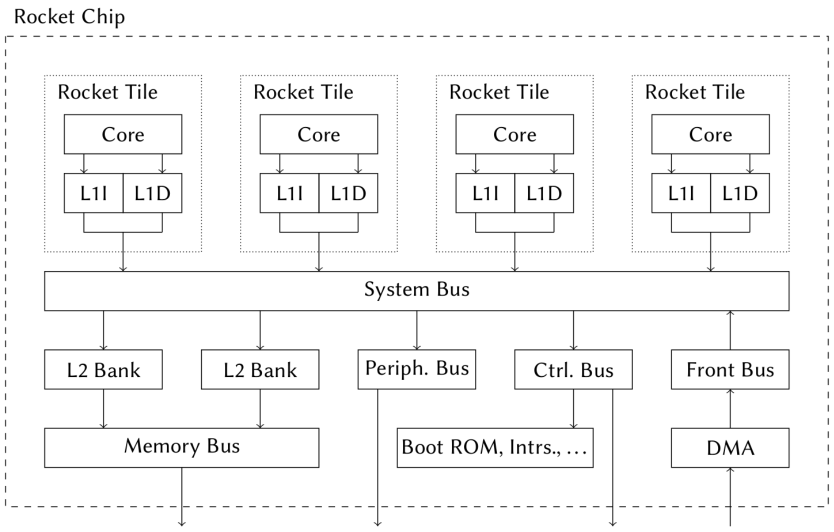


Fig. 2. The Rocket Chip [5] consists of multiple processor tiles, and devices connected through dedicated buses, like memory and peripherals. All parts of the chip communicate through the system bus

2.3. TileLink

TileLink is an on-chip interconnect standard and is used in Rocket Chip. It was developed with the motivation of creating an open standard, that is easy to implement, supports cache block motion and multiple cache layers, is reusable off-chip, and has high performance [37].

TileLink is able to connect any SoC component (agent) through links. [35, 9-12] Agents can implement master interfaces requesting memory access and operations and/or slave interfaces managing requests of their paired master interface. Links can comply with three different conformance levels:

– TileLink Uncached Lightweight (TL-UL)
– TileLink Uncached Heavyweight (TL-UH)
– TileLink Cached (TL-C)

The conformance level dictates how many channels a link includes and what type of operations are supported. TL-UL and TL-UH use two channels, one from master to slave and one from slave to master. TL-UL supports only simple single-word memory operations, to which TL-UH adds capabilities like burst accesses. TL-C further adds three new channels allowing coherent cache access support. [35, 7-8]

TL-UL uses five, TL-UH nine, and TL-C 21 different types of messages [35, 40]. A message consists of multiple fields carrying, for example, information about the sender and receiver, a payload, and other data, all depending on their type. To confirm complete transactions TileLink uses a ready-valid handshake mechanism.

3. Related Work

Worst-case execution time (WCET) estimation techniques bear the potential to analyse the timings of different hardware designs. Their data can be used to implicitly infer contention for shared resources. Some techniques could even provide explicit information about contention based on their internal intermediate analysis parts. Enabled comparisons can then guide decision making during hardware design. We now evaluate this prospect in more detail. WCET analyses aim to bound to maximal, i.e., worst-case execution of tasks to enable their use in mission or safety-critical real-time systems. WCET analyses encompass static, measurement-based and hybrid techniques. These are surveyed in [1]. Maiza et al. specifically analyse timing verification techniques for multicores [26]. Analytical approaches are mostly burdened by their inherent compute and/or preparation requirements. Especially approaches like [18], [10], [42], [31] being based on model-checking are restricted by their own complexity and scalability as indicated by their authors and also concluded in [26]. A different group of methods, like the multicore response time analysis (MRTA) framework [12] integrate their interference predictions into the scheduling and also achieve a certain flexibility in respect to supported hardware configurations. However, they all are afflicted with the need for creating and updating models. These processes then, could either be disproportionate in their cost or may not be able to consider small intricacies of a new, slightly optimized hardware design. Therefore they are ill suited for early stage agile hardware design and rapid testing. Hence they fulfill a very different purpose to Manatee. Other methods employ different forms of isolation as a direct means to limit the interferences on shared resources: Temporal isolation in the form of time division multiple access (TDMA) bus arbitration as employed in [23] or [43], and/or spacial isolation for shared memory (e.g. in [11][44]), represent such approaches. While often improving WCETs and their analysis, drawbacks of these techniques come in the form of lower average-case performance and higher energy consumption [26]. Moreover, they too, just by their very nature are unable to function in a hardware prototyping process which needs to support an exhaustive array of potential hardware and software configurations. In conclusion, current techniques of WCET estimation are not capable of accompanying an agile hardware development process in a sufficient manner for SoCs with real-time requirements. Manatee however aspires to do just that by compromising on the completeness, which regular WCET analysis provides, in turn gaining hardware independence, automation and speed.

For these required properties, better suited through faster evaluation are tools that concentrate on the actual measurement and interpretation of multicore interferences. Efforts to the likes of [22] first measure the interferences to later generate a worst possible co-runner. However, in alignment with their objective of analysing commercial off-the-shelf (COTS) hardware, they rely exclusively on platform-native performance monitor counts (PMCs). González et al. [27] also use platform native PMCs in conjunction with a measurement-based probabilistic WCET analysis to investigate interferences on embedded COTS. Lesage et al. even analyse the significance of a number of PMCs in relation to interferences in [24]. The evaluations of [29] provide further evidence for the benefits of the integration of hardware PMCs in general. We exploit these benefits even further by creating our own customised hardware counters to gather information of even higher relevancy. Apart from the aforementioned methods, noteworthy in the context of multicore interference are online detection methods which allow through recovery the preservation of the functionality of their systems. For example Esposito et al. [14] associate PMCs to previously offline defined thresholds. Here, again WCET analysis and/or extensive profiling is required while also following a different goal compared to this paper.

Conversely, there are established frameworks and theoretical methodologies pertaining to the design of hardware. Most literature (e.g. [8]) for hardware design and evaluation in general does not or only insufficiently integrate shared resource interference awareness. Specifically in the field of avionics there exists work for the software deployment on multicore hardware with reduced shared resource interference [17]. The focus of Girbal et al. [17] is however more directed to the software aspect and configuration of COTS hardware and does not support the development of new hardware, as we propose in this paper.

We also explore the different forms of practical tooling to support hardware design in the following. The already introduced gem5 simulator [7,25] is a framework supported through many contributors. It is open-source and widely used in academia and industry. The simulation infrastructure offers a substantial component library and can be extended with custom parts. Further, it provides the ability to test quickly and evaluate, and is therefore well suited for early prototyping [34]. However, as already suggested, the gem5 model is not cycle-accurate [13], especially the memory model seems to suffer from inaccuracies [9]. This is to the detriment of interference measurements. Manatee's chosen Chipyard tooling specifically avoids that through its utilisation of verilator and its memory simulation. The integration of FPGAs allows for even greater accuracy.

A number of other open-source tools are in their functionality to some degree comparable. MARSS (Micro Architectural and System Simulator) [32] expands on the core model of PTLsim [45], a simulator for microprocessors, adds models for system parts like caches, and couples all with the emulation capabilities of QEMU [6], a full-system emulation tool. This combination allows MARSS to switch between fast emulation and cycle-accurate simulation, where beneficial [32]. One of its drawbacks, however, is its limitation on x86 architectures. Further, lacking FPGA support MARSS is bound to much higher simulation times compared to Manatee's Chipyard integration.

Besides these open-source tools, proprietary prototyping frameworks exist as well. They generally come with the disadvantage of tight restrictions to certain architectures and parts. An example is Arm's DesignStart [2], offering a platform and packages to customise SoC designs. This includes IP cores, like Arm's Cortex-M0, but also regular peripherals and AMBA buses [38].

Chipyard [2] already introduced and further explored in the later requirements section is not plagued by the drawbacks of the different other tools summarized. Still, Chipyard itself does not include evaluation capabilites for shared resource contention.

Our needs and motivations are situated at the margins of disparate fields, yet the available solutions do not satisfy our requirements. In Manatee we create a hardware development flow with steady awareness of shared resource interferences in mind, unlike previous development techniques and tooling. We also incorporate multiple avenues of technical feedback by including functional, simulation-based and FPGA-based testing cycles (see Fig. 4). Not being able to adapt classical WCET analysis methods because of the also previously listed restrictions we rely on a small footprint measurement-based approach directly targeting interferences. By the virtue of the direct involvement in the hardware prototyping process, we can act independet of existing, rather limited PMCs and attach our own impermanent PMC hardware for the then needed measurements. This, of course, is advantageous in our specific case and separates from related works with PMCs.

4. Manatee: A Tool to Provide Multicore Interference Awareness

Here we describe the tool and framework Manatee which later is able to support research on and development of timing-analysable memory hierarchies for embedded multicore SoCs. We first introduce requirements this type of research demands then answer these requirements on a conceptual level. Implementation details later build on this concept and lead to the framework's realisation.

4.1. Requirements for Research on Timing Predictable Shared-memory Multicore Systems

To approach the objective of calculating tight WCET bounds for time-sensitive tasks in shared-memory multicore systems, the potential interferences on shared resources have to be identified and measured first. While such evaluations can be performed on existing hardware, potential new methods to prevent or restrict interferences require customisable hardware components. A system that enables the modification and enhancement of individual elements in the memory hierarchy should fulfill the following requirements:

– Customisable hardware to extend or modify elements of the memory hierarchy

[2] https://www.arm.com/resources/designstart

- Measurement of the overall performance and counting individual accesses on shared resources
- Independence of CPU architectures
- Scalable number of processor cores
- Hardware cost estimation of extensions and customisations
- Fast response on functional correctness of the implementation
- Fast and approximate evaluation of the simulated model
- Accurate full-system evaluation on an FPGA

These requirements are satisfied by Manatee, for which the Chipyard project provides a promising foundation. It is the predestined choice, since it is built around the open RISC-V ecosystem [28], and allows to customise or replace individual elements of the memory hierarchy. It further supports simulation and FPGA synthesis based on the same and identical code.

One concern while building this new framework is to not limit this given freedom by constraints coming from framework parts other than Chipyard. This is especially a defining quality for the hardware components tasked with making measurements as they are part of and directly influence the Chipyard hardware generation. They should not cause conflicts, for example, during parameter negotiation with other components. Also, they should be easily attachable and removable since they are only a tool for testing and are not supposed to be in released chip hardware. Their exact task is to count individual hardware accesses while the measuring itself should also not impact the workload execution.

In addition, the measured access counts need to be divided or grouped by criteria representing the most important details of the measured accesses to allow later analysis and evaluation. Results need to be visualised to facilitate interpretation and to provide a comparative overview of multiple tests.

The requirement for a ”Fast and approximate evaluation of the simulated model” effectively demands automation of the prototyping flow. Tasks, like the preparation of the workload, its compilation, and the simulation, need to be pipelined and executed in parallel. For workload content we chose the TACLe Benchmark collection [15]. It is comprised of 57 single-core benchmarks, which can be combined into multicore programs. The amount of possibilities requires the workload preparation itself also to be automated.

The following section presents the concept of a framework adhering to these formulated requirements.

4.2. Concept

A common objective of research on memory hierarchies for real-time systems is to reduce interferences on shared resources. From this, the main elements of the system under evaluation are derived: All units that control access to shared resources, like the peripheral bus, or the L2 cache, are of interest, as well as the private L1 caches that are connected to the shared system bus. In Fig. 3, these elements are shown below the processor cores, which are not of special interest for interference analysis. All accesses to shared resources that originate in the cores have to pass through the L1 instruction or data caches, which can control the communication. The prototyping flow from implementing a design of one or more specific parts of the memory hierarchy to code generation and simulation or evaluation is depicted in Fig. 4. Unit tests can provide fast checks of the functional correctness

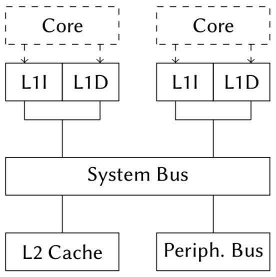


Fig. 3. Elements of interest to evaluate interferences in the memory hierarchy: private L1 caches, shared L2 caches, buses that connect shared resources, and the shared system bus itself

of the implemented or modified mechanisms. After passing these tests, Verilog code is generated, which can be simulated with Verilator to test the design with a set of benchmarks. The simulation provides fast feedback on the behaviour of the system, to compare different potential implementations before running the full evaluation of the synthesised bitstream on the FPGA. The evaluation of the design on the FPGA provides accurate timing measurements of the individual tasks, and a trace log of accesses on shared resources. These results allow to quantify the improvements of the implemented memory hierarchy modifications, and enable the detection of timing violations or forbidden interferences that should not occur. The possibility to connect a debugger to the simulation, as well as

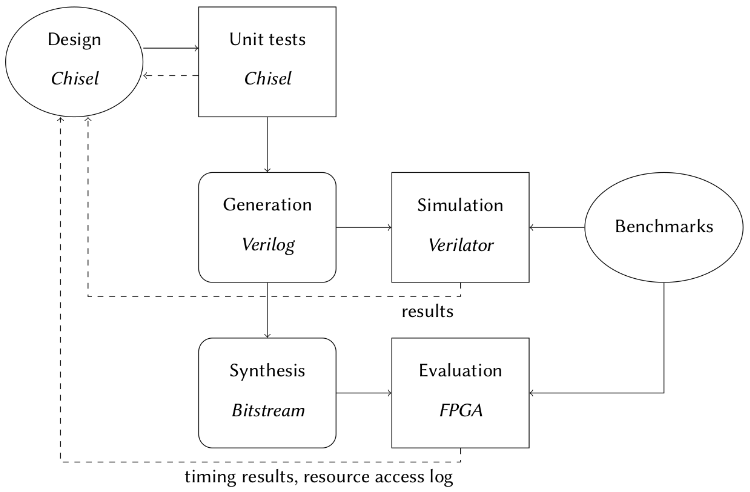


Fig. 4. Overview of the prototyping flow with Manatee. Results of the unit tests provide immediate feedback, which can be further tested in the Verilator simulation. The evaluation of the bitstream on the FPGA provides accurate timing results and logs of the resource accesses

to the system on the FPGA, facilitates the detection of implementation faults, and provides detailed insight into the behaviour of the system under specific circumstances when needed. With the feedback loop between the design and the simulation, available computational capabilities can be leveraged to compare numerous different design variations, to select a few designs of interest for the full evaluation of the FPGA.

4.3. Implementation of the Simulation Loop

This extended paper realises the previously envisioned concept [20] partially. Among the three possible feedback loops of the prototyping flow, we chose the simulation-centered one (see Fig. 5) to be the focus of our first steps forward. While the conceptual proto-

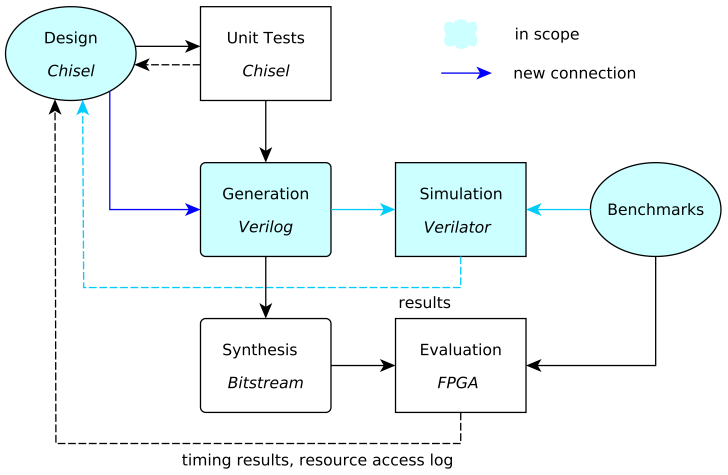


Fig. 5. The concept with a marked scope of this extension

typing flow showed only the main components of the framework (see Fig. 4) the more detailed view of Fig. 6 now includes all additional intermediate steps necessary for implementation. In the following, we describe the respective components:

The *measurement facility* is a hardware instrument to count or protocol any kind of traffic it detects. In the SoC design, it can be connected to the points where relevant shared resource accesses occur. Designed as a Chipyard configuration mixin[3] this component can be readily attached between most of the previously declared components of interest (see Fig. 3). The perhaps most relevant connection between the system bus and the L2 cache is included. In the standard configuration of a Chipyard SoC the L2 cache is the only shared memory and therefore represents the resource of the highest contention with most contenders traffic coming through the system bus.

[3] Mixins are a simple way in Scala to extend a class through the so called cake pattern. Chipyard and Rocket Chip leverage this mechanism to provide user friendly configurability.

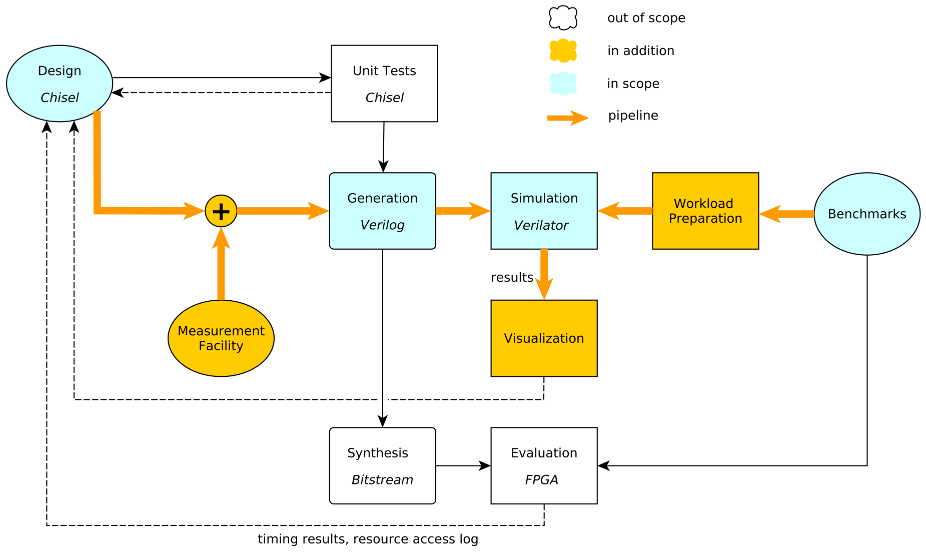


Fig. 6. Intermediate steps to the conceptual prototyping flow from Fig. 4

The *pipeline* connects all stages of the prototyping flow after the manual inclusion of the measurement facility (see Fig. 6). It automates the progression through all included steps to conform to the previously stated requirements. The expected execution times also made parallelisation necessary. Fig. 7 shows the different paths and functionalities the pipeline includes.

One step in this pipeline is the *preparation of multicore workloads*. This involves the combination of chosen benchmarks into a single program. The program assigns each of its included benchmarks to its own processor core resulting in parallel execution. Further, Manatee adds control capabilities to the program/workload to command the measurement facility later during measurement executions.

A *visualisation* module makes up the final step in a cycle of the prototyping flow. Its purpose is to show the results after the pipeline finishes its execution. This visualiser is realised as an interactive web application with extensive capabilities suited for navigating, analysing, and comparing the amounts of data produced by the measurements of Manatee.

Fig. 8 depicts the prototyping flow from a data-focused perspective also showing the control and read communications between the hardware module of the measurement facility and a workload. The most important features of the implementation are:

- Workload generation from any given folder containing suitable C programs
- Automatic assembly and processing of workloads for any number of cores
- Two different workload compilation methods [4]

[4] 1. With a libgloss port using the Berkeley Host-Target Interface (HTIF) https://github.com/ucb-bar/libgloss-htif/ (standard method of chipyard)
2. Following the pattern of the official riscv-tests repository https://github.com/riscv-software-src/riscv-tests/ (alternative for reference and in case of incompatibilities)

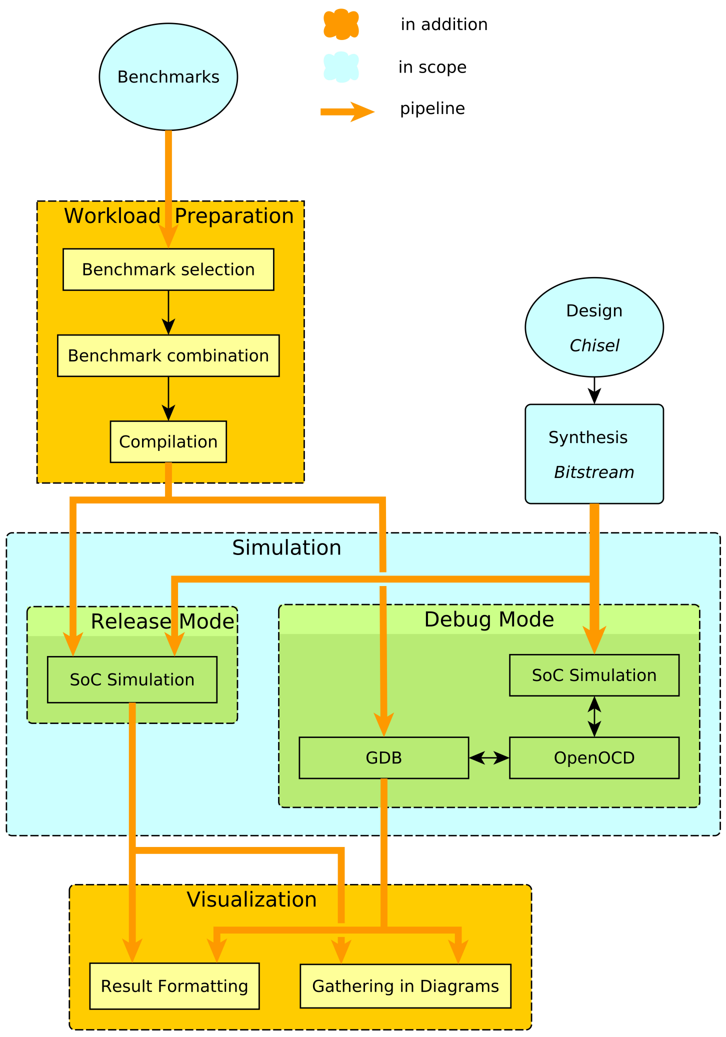


Fig. 7. An overview of the main pipeline parts and their connections

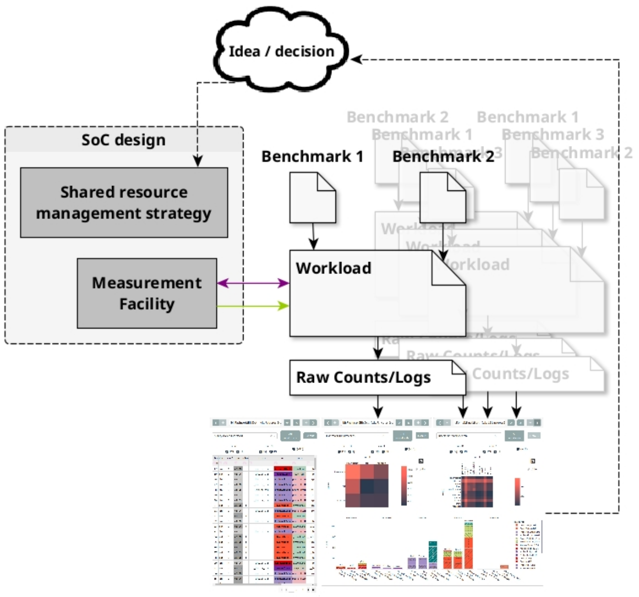


Fig. 8. The prototyping flow from a data-focused perspective

- Generation of any given Chipyard configuration with following simulations
- Option to mask cores, to not receive tasks
- Two different simulation methods (directly, and over GDB and OpenOCD)
- Processing of both detailed logs and regular access counts (measurements of the measurement facility)
- Automated decoding of recorded memory addresses with the help of gdb objdumps
- Interactive multi-modal visualisation of the measurement results

This implementation is freely available[5] and evaluated through the following section.

5. Evaluation

This section constitutes proof of the functionalities Manatee provides. At the same time, it is intended to serve as a reference for framework users investigating their hardware configuration. For this purpose, different smaller use cases are presented and solved with Manatee. These use cases are then described in the overarching context of the development loop for a shared resource management strategy. In addition we give some general insight into the L2 communication patterns of RISC-V programs with newlib/libgloss[6] on Rocket Chip cores. The section ends with a short discussion of its contents.

[5] https://es-augsburg.de/manatee
[6] RISC-V's libgloss port: https://github.com/ucb-bar/libgloss-htif/

5.1. Intended Use Cases

Filtering Benchmarks of Interest The different benchmarks TACLeBench [15] is comprised of vary widely in their resource requirements and other statistics. Manatee can provide an overview showing the cycle and message numbers in graphs. Fig. 9 is an example. This enables the user to find benchmarks best suited for their optimisation problem more easily.

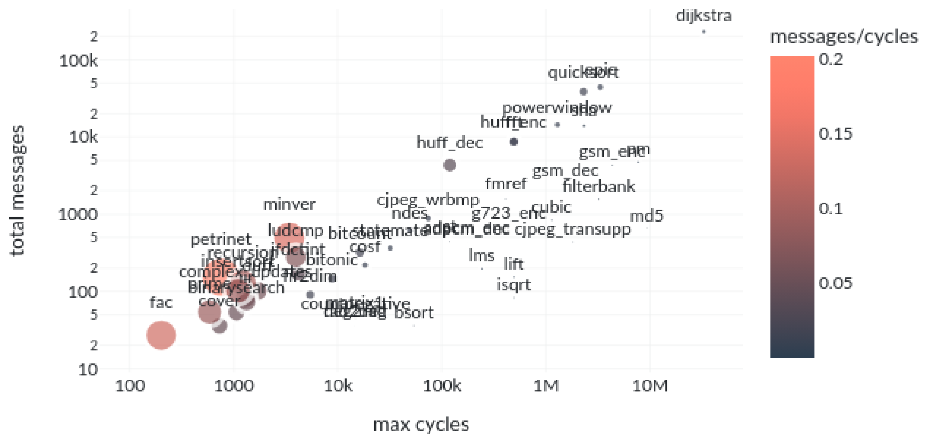


Fig. 9. A graph produced by Manatee showing the number of messages exchanged between the L2 cache and the L1 data and instruction caches over execution time for single benchmark workloads. Log mode is activated to better fit all workloads

Hover-text and readily available bar charts provide more detailed information, further supporting benchmark and workload selection by the user.

Filtering Workloads of Interest with Heatmaps Heatmaps can provide a very informative and quick overview over multiple workloads in dual-core setups. Manatee constructs these multicore workloads automatically in its workload preparation stage by combining the appropriate number of single core benchmarks (see section 4.3.). The following example compares two hardware designs, each featuring two big Rocket cores. The first design has L1 data caches with eight cache sets each, while L1 set sizes in the second design are quadrupled. Heatmaps of each selection show their performance and allow a rough comparison (see Fig. 10 and Fig. 11).

The scales and patterns of both heatmaps are very similar. Hovering over parts shows their number of L2 messages and execution times. The general difference of most workloads can be estimated to be around 20%. However, for example the heat of countnegative-deg2rad (seventh row in the last column) is noticeably different. Hovering already reveals a vastly more significant change compared to the rest of the workloads. This can then be further specified with the readily available bar charts.

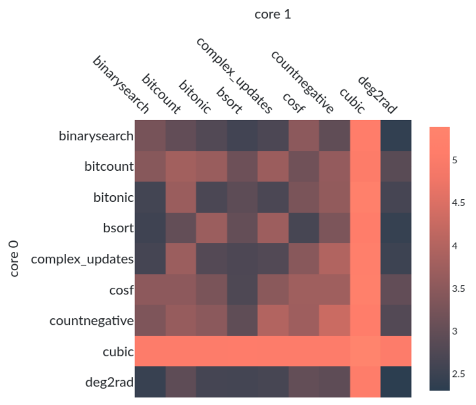


Fig. 10. The heatmap shows measurement results for a selected number of benchmark combinations. The heat is generated from the number of messages exchanged between the L2 cache and the L1 data caches of each core on a scale of log10. The used hardware design consists of two Rocket cores with eight L1 data cache sets

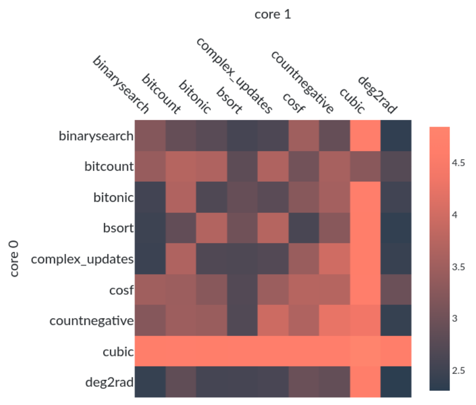


Fig. 11. The same visualisation setting and similar hardware. Here each core has 32 L1 data cache sets

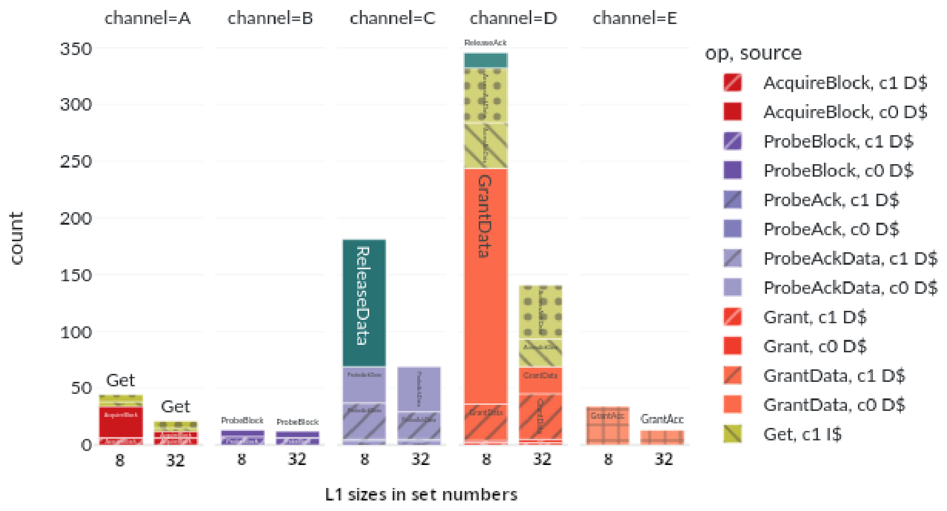


Fig. 12. Comparing the L2 cache accesses between two different hardware configurations for countnegative on the first and deg2rad on the second core

Fig. 12 depicts the bar chart of countnegative-deg2rad, which allows further investigation. The user can also conduct analysis in even more depth as described in Detailed Access Analysis to investigate changes like this.

Analysis of Recorded L2 Accesses This section presents examples of measured message counts and provides short analyses based on typical TileLink behaviour. Effectively, the user can use this information to estimate the potential delay time a benchmark can cause on the tested hardware.

L1 Data Cache Traffic Fig. 13 shows the messages between the L2 cache and the L1 data cache of one core in a dual-core setup during benchmark execution. The colouring marks three flows and their progression:

– Red colours: AcquireBlock $\longrightarrow$ Grant(very small)/GrantData $\longrightarrow$ GrantAck
– Green colours: ReleaseData $\longrightarrow$ ReleaseAck
– Purple colours: ProbeBlock $\longrightarrow$ ProbeAck/ProbeAckData(also very small)

These flows typically interact as shown in Fig. 14. Since AcquireBlock, ReleaseAck, and ProbeBlock are only single-beat[7] messages, we can deduce the number of their respective flow executions directly from their numbers. The L1 data cache starts with the AcquireBlock request. From the ratio between the number of AcquireBlocks (134) and ReleaseAcks (39), we can infer that the L1 data cache follows this request immediately up with a ReleaseData message in nearly 30% of the cases. The reason for these releases is often capacity conflict [35, 65] (which is to be expected from the small number of

[7] beat = clock cycle

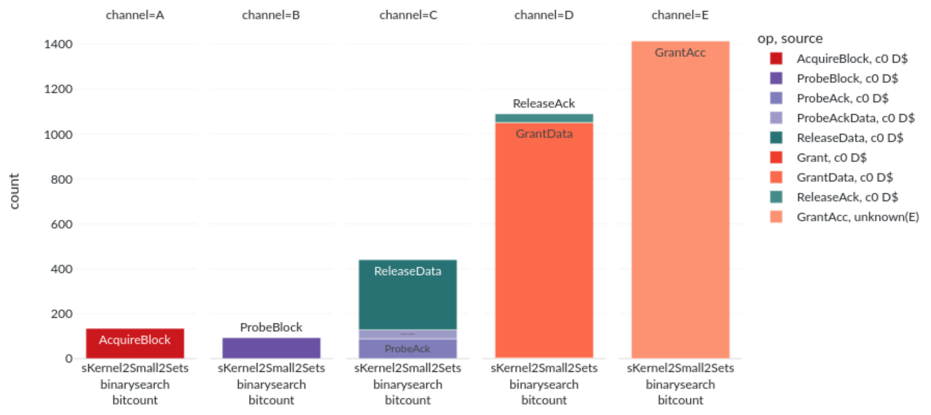


Fig. 13. Hardware: Two small Rocket cores, each with only 2 sets in their L1 caches; Workload: core 1: binarysearch, core 2: bitcount; The graph shows messages between L2 cache and the L1 data cache of the first core

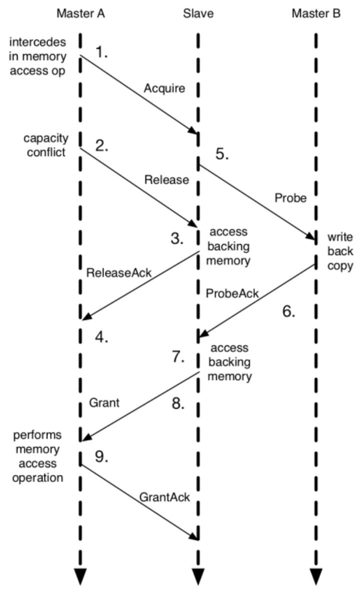


Fig. 14. Taken from [35, 73]. This graph shows one of the most common interactions between TL-C level message flows. In our case the masters are L1 caches and the slave is the L2 cache

L1 cache sets). Once the L2 cache receives a ReleaseData block, it confirms with a ReleaseAck. The ratio between ReleaseData (312) and ReleaseAcks (39) confirms the block length of a ReleaseData block at eight beats. The L2 cache also wants to grant the data and permission from the original AcquireBlock request. However, if it previously granted this permission to another master, it first needs to receive it back. Using the communication data between the L1 data cache of the second core we can find out how often this was the case. Fig. 15 counts 87 ProbeBlocks meaning that in nearly two out of three times, the L2

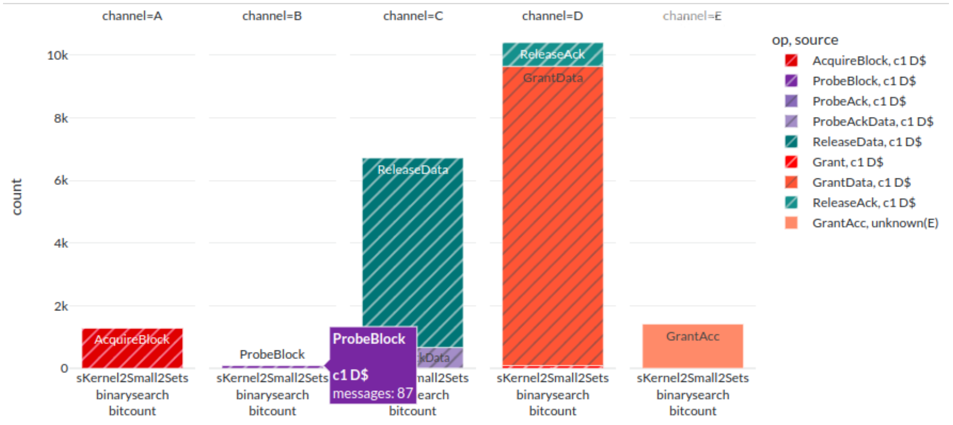


Fig. 15. Same data set as Fig. 13; In contrast, this figure shows the counts of bitcount executed on the second core

cache had to first get the permissions back from the L1 data cache of core two. Finally, closing the red flow, L1 sends the GrantAck message to the L2 cache. Here needs to be noted that the total displayed number in both Fig. 13 and 15 is the total of all GrantAcks recorded. As the GrantAck message does not have a source identifier, it is not assigned to a specific master interface (in this case, the L1 data cache) when recorded. However, its number (1414) matches the sum of AcquireBlock requests of both L1 data caches (134 and 1280).

L1 Instruction Cache Traffic The message flow between an L1 instruction cache and the L2 cache is simpler. Fig. 16 depicts the message counts related to the instruction cache of the first core in the previously examined dual-core setup. The single flow necessary for the instruction cache consists of a Get request from the instruction cache and AccessAckData answers from the L2 cache. With counts of 120 AccessAckData messages and 15 Get messages, the AccessAckData block size of eight beats can be confirmed.

Delay Potential The in previous sections examined message counts can be used as a basis to estimate in how much a benchmark was negatively influenced by its co-runners or what its delay potential affecting other benchmarks might be.

Another very obvious ratio is the total message number to execution time since the L2 cache can only receive one message at the same tick.

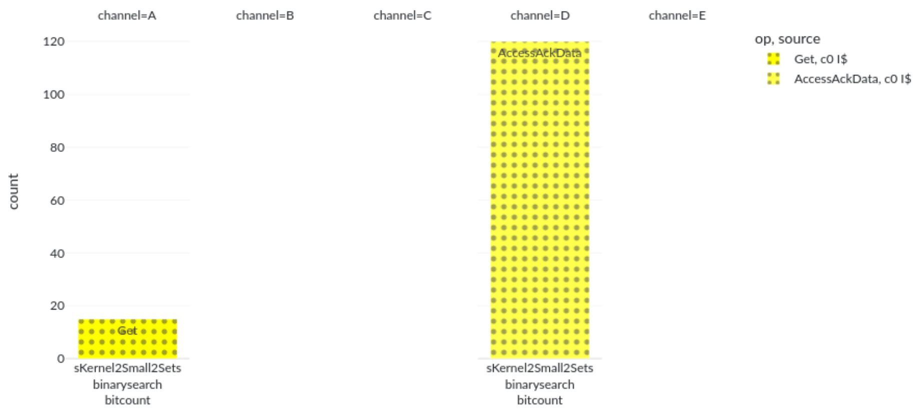


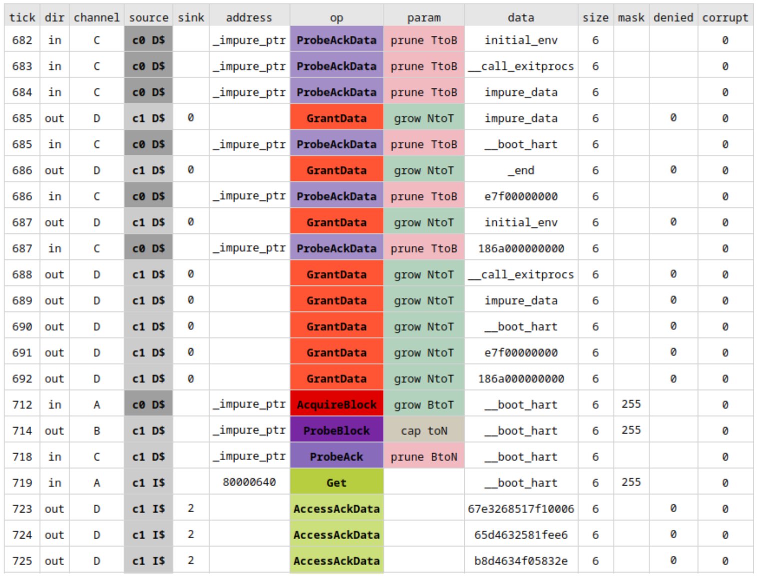
Fig. 16. Same data set as Fig. 13 and 15; Here instead of the counts of an L1 data cache L1 instruction cache counts are selected

One already hinted indicator can be the number of ProbeBlocks (to different cores) or the ratio of ProbeBlocks (to different cores) to AcquireBlocks (from this core). This follows from the previously explained flow interaction, also shown in Fig. 14. This can cause even further delays through edge cases initiated through circumstances like network delays or a concurrent Release and ProbeBlock [35, 72-74].

Detailed Access Analysis In some cases, an analysis, as presented above, raises new questions or is inconclusive. The user can then execute the workloads of interest with the message logger instead of the regular counter to gain more information. After this, the user can view all messages in detail. Fig. 17 shows part of the recorded messages caused by a binarysearch-insertsort workload as an example.

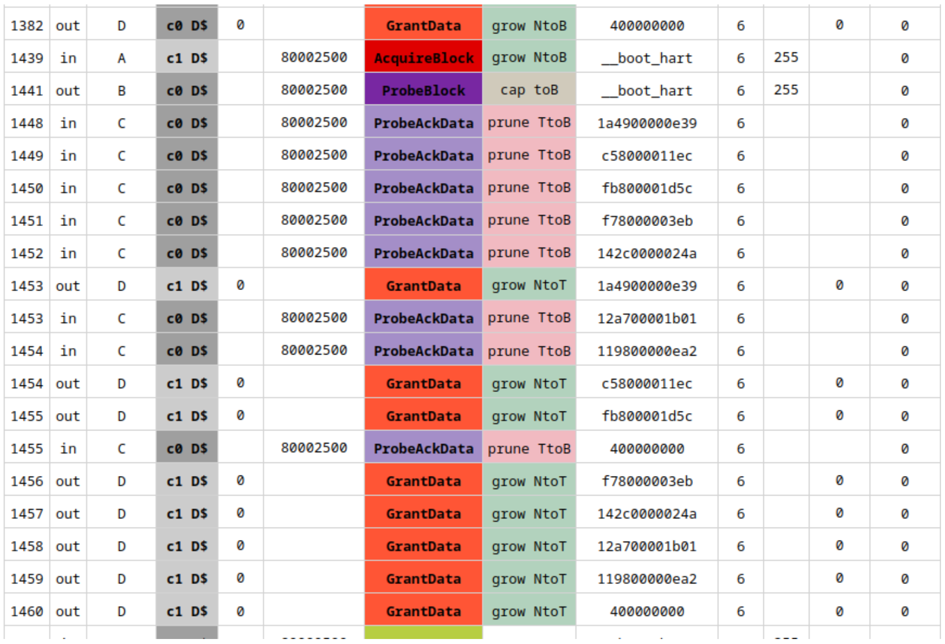
This figure also shows the decoder resolving some addresses with the help of an objdump symbol-table. In this case, translated addresses like *_impure_ptr*, *initial_env* and *__call_exitprocess* are part of the newlib library. The address and data of instruction cache-related messages (here in greenish-yellow) are unresolved but can be manually searched in the objdump file, which the pipeline generated on execution. This reveals *80000640* to be part of the memory where insertsort's main instructions are stored. The AccessAccData messages, which the L2 cache responds with, contain these instruction codes.

Fig. 18 shows another part of the already introduced table. The first AcquireBlock message in this extract requests a cache block for the second core currently held by the first core. Followed by the L2 cache getting back the permissions from the first core and granting them to the second core. Not included in the figure, the table displays this behaviour multiple times repeated and also reversed. Searching the objdump shows the locations (see Fig. 19). Binarysearch occupies the memory addresses from *0x800024c0* to *0x80002537* and insertsort holds the addresses from *0x80002538* to *0x80002563*. The block size of the L2 cache is 64 bytes. This means the competing cache block accesses result from an overlap in the cache block, which has the content from *0x80002500* to *0x80002539*.



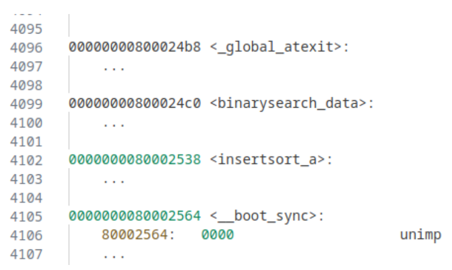
tick	dir	channel	source	sink	address	op	param	data	size	mask	denied	corrupt
682	in	C	c0 D$		_impure_ptr	ProbeAckData	prune TtoB	initial_env	6			0
683	in	C	c0 D$		_impure_ptr	ProbeAckData	prune TtoB	__call_exitprocs	6			0
684	in	C	c0 D$		_impure_ptr	ProbeAckData	prune TtoB	impure_data	6			0
685	out	D	c1 D$	0		GrantData	grow NtoT	impure_data	6		0	0
685	in	C	c0 D$		_impure_ptr	ProbeAckData	prune TtoB	__boot_hart	6			0
686	out	D	c1 D$	0		GrantData	grow NtoT	_end	6		0	0
686	in	C	c0 D$		_impure_ptr	ProbeAckData	prune TtoB	e7f00000000	6			0
687	out	D	c1 D$	0		GrantData	grow NtoT	initial_env	6		0	0
687	in	C	c0 D$		_impure_ptr	ProbeAckData	prune TtoB	186a000000000	6			0
688	out	D	c1 D$	0		GrantData	grow NtoT	__call_exitprocs	6		0	0
689	out	D	c1 D$	0		GrantData	grow NtoT	impure_data	6		0	0
690	out	D	c1 D$	0		GrantData	grow NtoT	__boot_hart	6		0	0
691	out	D	c1 D$	0		GrantData	grow NtoT	e7f00000000	6		0	0
692	out	D	c1 D$	0		GrantData	grow NtoT	186a000000000	6		0	0
712	in	A	c0 D$		_impure_ptr	AcquireBlock	grow BtoT	__boot_hart	6	255		0
714	out	B	c1 D$		_impure_ptr	ProbeBlock	cap toN	__boot_hart	6	255		0
718	in	C	c1 D$		_impure_ptr	ProbeAck	prune BtoN	__boot_hart	6			0
719	in	A	c1 I$		80000640	Get		__boot_hart	6	255		0
723	out	D	c1 I$	2		AccessAckData		67e3268517f10006	6		0	0
724	out	D	c1 I$	2		AccessAckData		65d4632581fee6	6		0	0
725	out	D	c1 I$	2		AccessAckData		b8d4634f05832e	6		0	0

Fig. 17. Part of the message table of a binarysearch-insertsort workload. The *source* column names the communication partner of the L2 cache, e.g. *c0 D$*: L1 data cache of the first core. The term ”source” refers to the master role of the the respective communication partner. The *dir* column shows the direction of the message from the point of view of the slave (here the L2 cache)



1382	out	D	c0 D$	0		GrantData	grow NtoB	400000000	6		0	0
1439	in	A	c1 D$		80002500	AcquireBlock	grow NtoB	__boot_hart	6	255		0
1441	out	B	c0 D$		80002500	ProbeBlock	cap toB	__boot_hart	6	255		0
1448	in	C	c0 D$		80002500	ProbeAckData	prune TtoB	1a4900000e39	6			0
1449	in	C	c0 D$		80002500	ProbeAckData	prune TtoB	c58000011ec	6			0
1450	in	C	c0 D$		80002500	ProbeAckData	prune TtoB	fb800001d5c	6			0
1451	in	C	c0 D$		80002500	ProbeAckData	prune TtoB	f78000003eb	6			0
1452	in	C	c0 D$		80002500	ProbeAckData	prune TtoB	142c0000024a	6			0
1453	out	D	c1 D$	0		GrantData	grow NtoT	1a4900000e39	6		0	0
1453	in	C	c0 D$		80002500	ProbeAckData	prune TtoB	12a700001b01	6			0
1454	in	C	c0 D$		80002500	ProbeAckData	prune TtoB	119800000ea2	6			0
1454	out	D	c1 D$	0		GrantData	grow NtoT	c58000011ec	6		0	0
1455	out	D	c1 D$	0		GrantData	grow NtoT	fb800001d5c	6		0	0
1455	in	C	c0 D$		80002500	ProbeAckData	prune TtoB	400000000	6			0
1456	out	D	c1 D$	0		GrantData	grow NtoT	f78000003eb	6		0	0
1457	out	D	c1 D$	0		GrantData	grow NtoT	142c0000024a	6		0	0
1458	out	D	c1 D$	0		GrantData	grow NtoT	12a700001b01	6		0	0
1459	out	D	c1 D$	0		GrantData	grow NtoT	119800000ea2	6		0	0
1460	out	D	c1 D$	0		GrantData	grow NtoT	400000000	6		0	0

Fig. 18. Another part of the message table of a binarysearch-insertsort workload



```
00000000800024b8 <_global_atexit>:
    ...

00000000800024c0 <binarysearch_data>:
    ...

0000000080002538 <insertsort_a>:
    ...

0000000080002564 <__boot_sync>:
    80002564:   0000                    unimp
    ...
```

Fig. 19. Part of the objdump of the referenced binarysearch-insertsort workload. binarysearch_data is an array of structs of the binarysearch benchmark. insertsort_a is a global array of integers of the insertsort benchmark

Comparing Shared Resource Management Strategies Manatee can be applied in an agile development process to frequently test new resource management strategies. Here, two small Rocket cores with L1 data and instruction caches of 2 sets will serve as the base configuration. The goal is to improve this hardware configuration.

In the first step, benchmarks for testing can be selected as described in the Filtering Benchmarks of Interest section. Depending on time and resource constraints, all benchmarks can also be used unfiltered. For this use case, all kernel benchmarks were filtered first through a single small Rocket core configuration. Benchmarks that the small Rocket core was not able to execute and benchmarks with long simulation times were then excluded.

After adding the measurement facility's mixin to the base Chipyard configuration the pipeline can be invoked again.

A heatmap (or scatter plot if the number of cores is not two) of all tested workloads gives now a first indication of the performance of individual cores depending on their task (see Fig. 20). Selecting different L2 communication partners gives further insight. If, for example, only one core is selected, resulting inconsistencies in the heatmap pattern can hint at different influences of different co-runners (see Fig. 21).

Based on this first data, workloads that, for example, require a lot of L2 communication can be investigated further as demonstrated in the Analysis of Recorded L2 Accesses and Detailed Access Analysis sections. The analysis results can give hints towards improving, for example, the shared resource management strategy.

Once a new experimental strategy is implemented it can be tested again. A comparison to the previous results shows the effects of the last development iteration in context. For demonstration purposes, the competing resource management strategy will be to double the number of sets in each cache. The new experimental hardware configuration can be evaluated in the same way as before. The visualiser allows a direct comparison, where a very significant decrease in total messages is noticeable (see Fig. 22).

For the next development iteration, new shared resource management strategies will be simulated by using big Rocket cores instead of the previous small ones. Fig. 23 shows the comparison of two workloads. Other not depicted workloads show the same trend. This leads to the conclusion that the last strategy is even better than the previous one.

This process can be repeated until testing results are satisfactory.

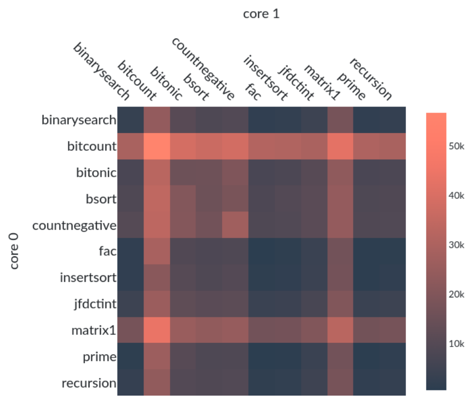


Fig. 20. Hardware: Two Small Rocket cores, each has L1 caches with only two sets. The data is from a selection of kernel benchmarks

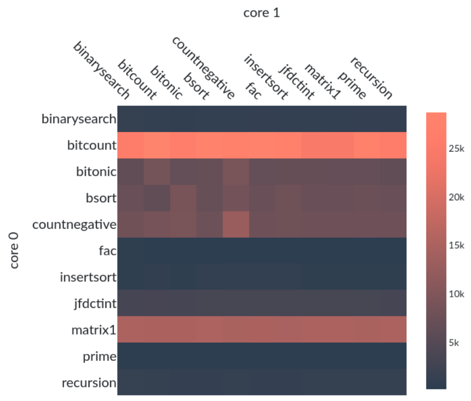


Fig. 21. Setup is the same as in Fig. 20, but only counts of messages from and to the first core (core 0) are shown

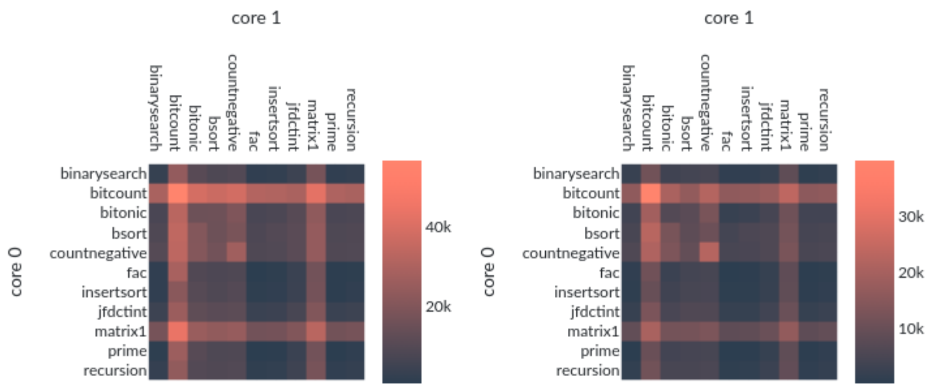


Fig. 22. Left setup is the same as in Fig. 20. The right setup has double the amount of cache sets (Notice the different heatmap scales on the right of each graph)

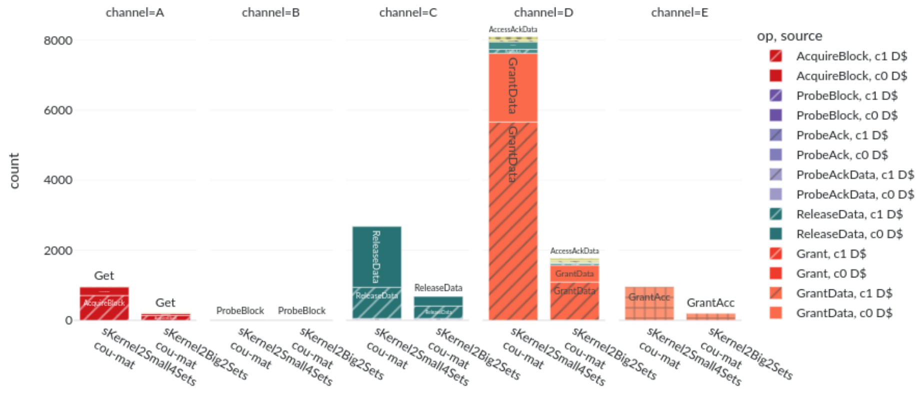


Fig. 23. Here, the visualiser pairs the message counts of different hardware configurations per channel to facilitate their comparison. The left setup is the same as the improvement of the previous iteration. The right setup has big Rocket cores but only half the number of cache sets

5.2. Additional Example Results

On Big Rocket Cores Big Rocket Cores and their preconfigured L1 caches are for most TACLe benchmarks to a degree sufficient where they only very rarely need to access the L2 cache. This leads to results where L2 access counts are dominated by regular operations of newlib/libgloss. In these cases, the recorded access counts can no longer provide sufficient performance indications. As an example the bitcount benchmark is shown (see Fig. 24). Here operations of newlib/libgloss or similar dominate the counts to a degree where a core without task has a nearly equal amount of L2 accesses.

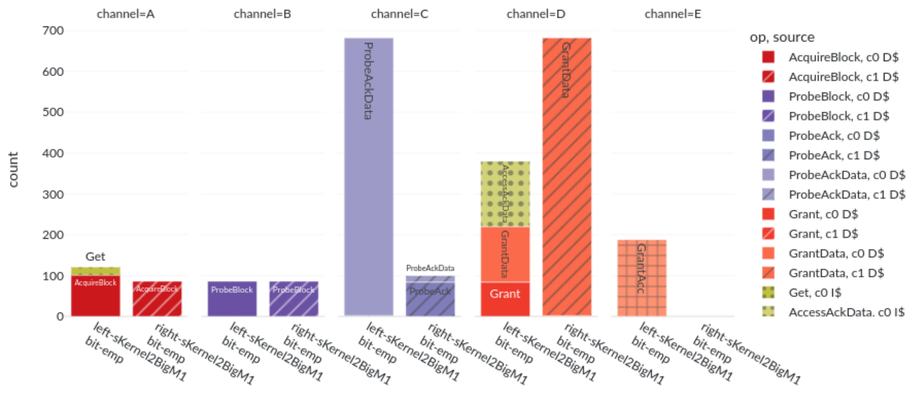


Fig. 24. Comparing the L2 cache accesses between two different cores. The workload consists of the bitcount benchmark running on the first core. The second core does not have a task assigned

Debug Mode As mentioned in a previous section, measurements done in debug mode showed differences to measurements from release mode. Fig. 25 shows an example.

The Impact of Additional Cores Fig. 26 shows the impact of doubling the core number two times on the benchmark of the first core. On each new core an additional task is run.

Between L2 Cache and Memory Bus Fig. 27 shows part of a workload-group-card, a vertical ordering section of the visualiser, with data of measurements conducted between L2 cache and memory bus loaded. For demonstration purposes, the L2 cache size has been reduced to 32kB to force continuous loads from memory.

5.3. Discussion

The use cases and the connected results gave a short introduction to the capabilities of Manatee. Especially the sections analysing the bar chart and the detailed table showed

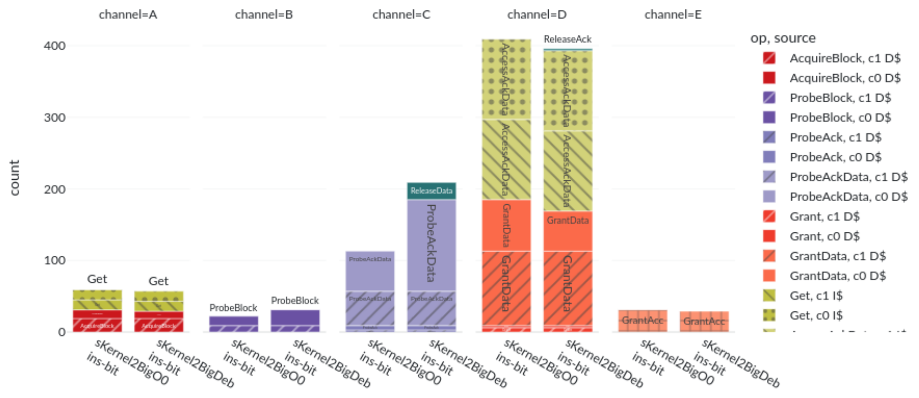


Fig. 25. Both sides use the same hardware simulator and the same workload code. However, the right side was executed in debug mode while the left side was executed regularly

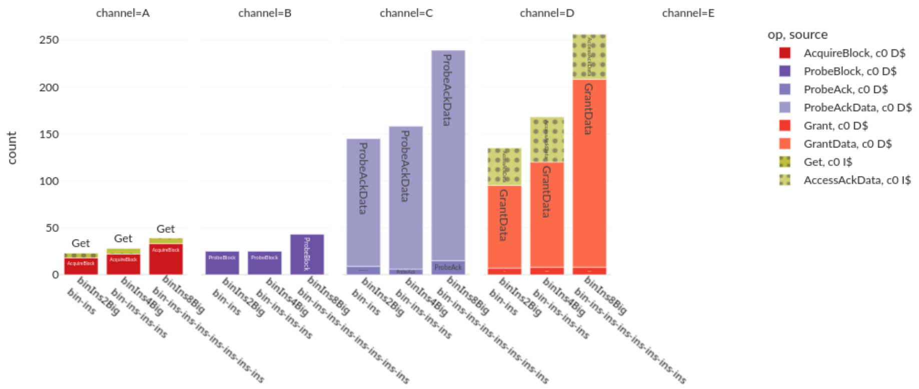


Fig. 26. The left bars of each channel's section represent dual-core workloads, the middle quad-core, and the right bars octo-cores. On the first core runs binarysearch while the tasks of the remaining cores consist of insertsorts. Each bar shows only messages associated with the first core

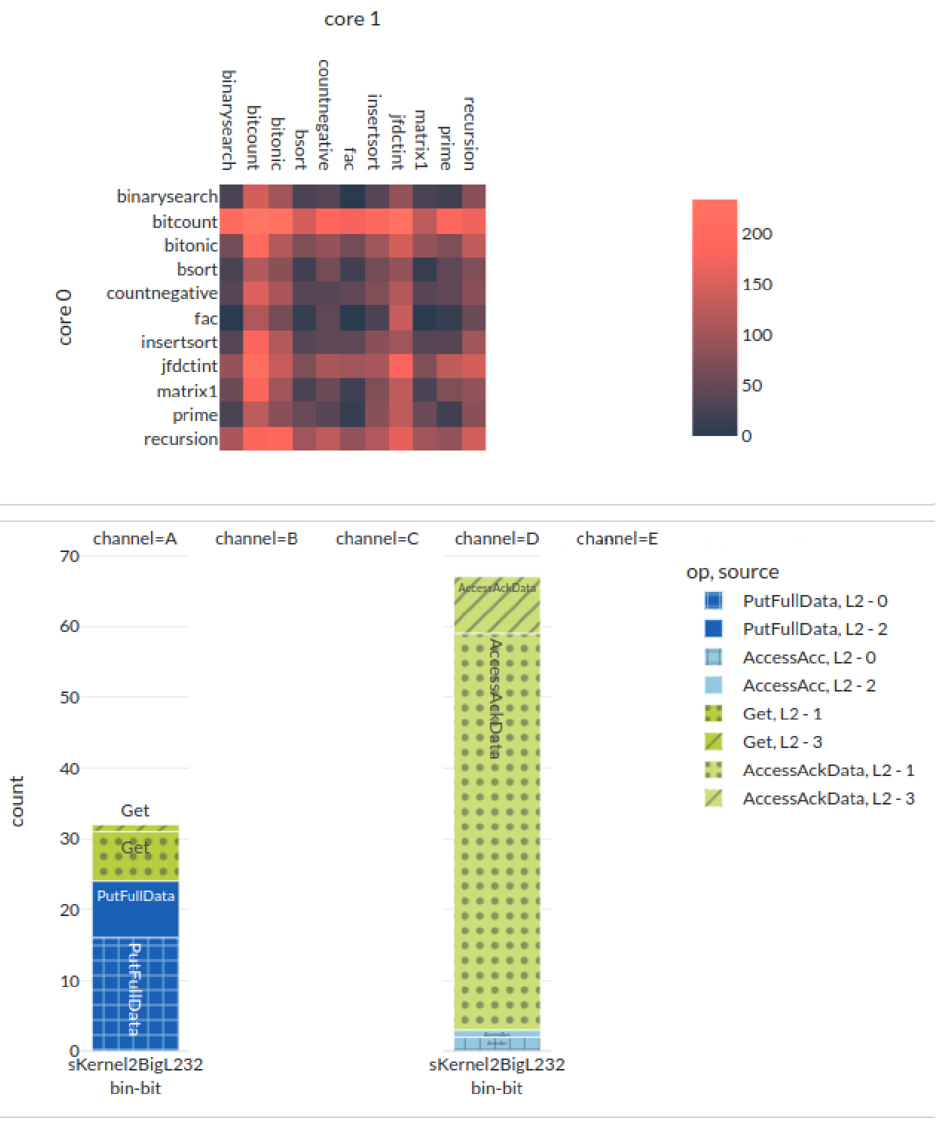


Fig. 27. On the top, a heatmap shows the message counts of workloads made up of a selection of kernel benchmarks. The lower part depicts a bar chart of the workload binarysearch-bitcount. Here the measurement facility monitored the L2 - memory bus connection instead of the system bus - L2 connection

the value of the measured bus traffic information. To be considered, however, is the simulation time. The, in the previous section, shown workloads all consisted of relatively small benchmarks so their simulation time was always short. But some other benchmarks of TACLeBench require more than $1 * 10^8$ cycles, which translates to more than ten hours of runtime even on capable systems. For chips with more cores, this effect gets expectedly worse. The wide range of benchmarks available in TACLeBench and the parallelised pipeline alleviate this issue somewhat. The conceptually already included addition of FPGA support will accelerate measurements even further and solve this limitation.

The analysis results gave a first idea of what to expect and also small interpretations of measurements. These interpretations were limited and partly superficial, as a closer exploration would have been out of the scope of this paper. A more thorough inspection of the workloads in their assembly code and their recorded message logs could reveal more relevant information.

During test data acquisition, the pipeline provided the necessary automation and parallel execution, reducing the amount of human work immensely. As part of this, the workload generator enabled workloads to be generated from any combination of benchmarks. During and after, the visualiser helped navigate the results and draw conclusions. All presented measurement graphs were created directly by the visualiser. The visualiser itself is readily available online[8] as a web application with all depicted and more evaluation data included.

As the evaluation examplified and documented, Manatee will provide a researcher with quick and continuous feedback through a hardware/software design process. This feedback includes multiple different indications and characteristics of interferences starting with identifying problematic software parts ending in the isolation of specific low level instructions. Always maintaining easy comparability and general comfort through the visualiser Manatee will further enable detailed evaluation for the current hardware design iteration. Through this, the potential of minimising multicore interferences already during the first design steps is greatly improved and should have significant likelyhood to affect the actual WCET in a beneficial manner. Further the interference measurements can be used in conjunction with WCET analysis methods as for example González et al. demonstrate in [27]. Crossreferencing then also can provide context and explain WCET analysis results.

6. Conclusion and Future Work

This paper described the design of the prototyping and evaluation framework Manatee to research on memory hierarchies, for getting closer to the overall objective of enabling high-performance multicore processors in embedded real-time systems. Manatee is built upon existing open-source projects around the RISC-V architecture, connecting the different tools together. It integrates all the required steps to automatically generate the Verilog code, compile and run the simulation, to synthesise the bitstream and program the FPGA with it, and to run the evaluation. We additionally provided an implementation of the largest part of the framework. This enabled tests and a subsequent assessment based on the use case of shared resource management strategy evaluation. Additional recorded

[8] https://es-augsburg.de/manatee

measurements gave some insight into access patterns to be expected from different factors. Among these are operations of newlib/libgloss, different-sized processor cores, and more.

This implementation of Manatee is now able to give the unprecedented opportunity to any embedded system developer and researcher to bring multicore designs to even critical use cases by providing fine-grained information about any possible processor core interference.

Future work is adding and integrating the FPGA and Chisel test cycles into this framework. Since the FPGA cycle overlaps the simulation cycle, large parts are already implemented. IO binders for suitable FPGAs are already available in the Chipyard repository.

As RISC-V continues to gain relevancy, Chipyard and its components are also very actively improved upon. This charges the potential of Manatee to support the future development and evaluation of successful shared resource management strategies for memory hierarchies in embedded systems.

Acknowledgments. This work is partially supported by the CERCIRAS COST Action no. CA19135 funded by COST.

References

1. Abella, J., Hernandez, C., Quiñones, E., Cazorla, F.J., Conmy, P.R., Azkarate-askasua, M., Perez, J., Mezzetti, E., Vardanega, T.: Wcet analysis methods: Pitfalls and challenges on their trustworthiness. In: 10th IEEE International Symposium on Industrial Embedded Systems (SIES). pp. 1–10 (2015)
2. Amid, A., Biancolin, D., Gonzalez, A., Grubb, D., Karandikar, S., Liew, H., Magyar, A., Mao, H., Ou, A., Pemberton, N., Rigge, P., Schmidt, C., Wright, J., Zhao, J., Shao, Y.S., Asanović, K., Nikolić, B.: Chipyard: Integrated design, simulation, and implementation framework for custom socs. IEEE Micro 40(4), 10–21 (2020)
3. Amslinger, R., Piatka, C., Haas, F., Weis, S., Ungerer, T., Altmeyer, S.: Hardware multiversioning for fail-operational multithreaded applications. In: 2020 IEEE 32nd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD). pp. 20–27 (2020)
4. Amslinger, R., Weis, S., Piatka, C., Haas, F., Ungerer, T.: Redundant execution on heterogeneous multi-cores utilizing transactional memory. In: Berekovic, M., Buchty, R., Hamann, H., Koch, D., Pionteck, T. (eds.) Architecture of Computing Systems – ARCS 2018. pp. 155–167. Springer International Publishing, Cham (2018)
5. Asanović, K., Avizienis, R., Bachrach, J., Beamer, S., Biancolin, D., Celio, C., Cook, H., Dabbelt, D., Hauser, J., Izraelevitz, A., Karandikar, S., Keller, B., Kim, D., Koenig, J., Lee, Y., Love, E., Maas, M., Magyar, A., Mao, H., Moreto, M., Ou, A., Patterson, D.A., Richards, B., Schmidt, C., Twigg, S., Vo, H., Waterman, A.: The rocket chip generator. Tech. Rep. UCB/EECS-2016-17, EECS Department, University of California, Berkeley (Apr 2016), http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.html
6. Bellard, F.: Qemu, a fast and portable dynamic translator. In: Proceedings of the Annual Conference on USENIX Annual Technical Conference. p. 41. ATEC '05, USENIX Association, USA (2005)
7. Binkert, N., Beckmann, B., Black, G., Reinhardt, S.K., Saidi, A., Basu, A., Hestness, J., Hower, D.R., Krishna, T., Sardashti, S., et al.: The gem5 simulator. ACM SIGARCH computer architecture news 39(2), 1–7 (2011)

8. Buchenrieder, K.: Rapid prototyping of embedded hardware/software systems. Design Automation for Embedded Systems 5, 215–221 (2000)
9. Butko, A., Garibotti, R., Ost, L., Sassatelli, G.: Accuracy evaluation of gem5 simulator system. In: 7th International workshop on reconfigurable and communication-centric systems-on-chip (ReCoSoC). pp. 1–7. IEEE (2012)
10. Dalsgaard, A.E., Olesen, M.C., Toft, M., Hansen, R.R., Larsen, K.G.: METAMOC: Modular Execution Time Analysis using Model Checking. In: Lisper, B. (ed.) 10th International Workshop on Worst-Case Execution Time Analysis (WCET 2010). Open Access Series in Informatics (OASIcs), vol. 15, pp. 113–123. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2010), `https://drops.dagstuhl.de/entities/document/10.4230/OASIcs.WCET.2010.113`
11. Dasari, D., Nelis, V., Akesson, B.: A framework for memory contention analysis in multi-core platforms. Real-Time Systems 52, 272–322 (2016)
12. Davis, R.I., Altmeyer, S., Indrusiak, L.S., Maiza, C., Nelis, V., Reineke, J.: An extensible framework for multicore response time analysis. Real-Time Systems 54, 607–661 (2018)
13. Elsasser, W., Nikoleris, N.: Memory controller updates for new dram technologies, nvm interfaces and flexible memory topologies (May 2020), `https://www.gem5.org/2020/05/27/memory-controller.html`
14. Esposito, S., Violante, M., Sozzi, M., Terrone, M., Traversone, M.: A novel method for online detection of faults affecting execution-time in multicore-based systems. ACM Trans. Embed. Comput. Syst. 16(4) (May 2017), `https://doi.org/10.1145/3063313`
15. Falk, H., Altmeyer, S., Hellinckx, P., Lisper, B., Puffitsch, W., Rochange, C., Schoeberl, M., Sørensen, R.B., Wägemann, P., Wegener, S.: TACLeBench: A benchmark collection to support worst-case execution time research. In: Schoeberl, M. (ed.) 16th International Workshop on Worst-Case Execution Time Analysis (WCET 2016). OpenAccess Series in Informatics (OASIcs), vol. 55, pp. 2:1–2:10. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2016)
16. Freitag, J., Uhrig, S., Ungerer, T.: Virtual Timing Isolation for Mixed-Criticality Systems. In: Altmeyer, S. (ed.) 30th Euromicro Conference on Real-Time Systems (ECRTS 2018). Leibniz International Proceedings in Informatics (LIPIcs), vol. 106, pp. 13:1–13:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2018), `https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ECRTS.2018.13`
17. Girbal, S., Pérez, D.G., Le Rhun, J., Faugère, M., Pagetti, C., Durrieu, G.: A complete toolchain for an interference-free deployment of avionic applications on multi-core systems. In: 2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC). pp. 7A2–1–7A2–14 (2015)
18. Gustavsson, A., Ermedahl, A., Lisper, B., Pettersson, P.: Towards wcet analysis of multicore architectures using uppaal. In: 10th international workshop on worst-case execution time analysis (WCET 2010). Schloss-Dagstuhl-Leibniz Zentrum für Informatik (2010)
19. Haas, F.: Fault-tolerant Execution of Parallel Applications on x86 Multi-core Processors with Hardware Transactional Memory. doctoralthesis, Universität Augsburg (2019)
20. Haas, F., Altmeyer, S.: A prototyping and evaluation framework for research on timing-analysable memory hierarchies for embedded multicore socs. In: CEUR Workshop Proceedings. vol. 3145 (2021)
21. Haas, F., Weis, S., Ungerer, T., Pokam, G., Wu, Y.: Fault-tolerant execution on cots multi-core processors with hardware transactional memory support. Lecture Notes in Computer Science 10172, 16 – 30 (2017)
22. Iorga, D., Sorensen, T., Wickerson, J., Donaldson, A.F.: Slow and steady: Measuring and tuning multicore interference. In: 2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS). pp. 200–212. IEEE (2020)
23. Kelter, T., Falk, H., Marwedel, P., Chattopadhyay, S., Roychoudhury, A.: Static analysis of multi-core tdma resource arbitration delays. Real-Time Systems 50, 185–229 (2014)

24. Lesage, B., Griffin, D., Bate, I., Soboczenski, F.: Exploring and understanding multicore interference from observable factors. In: Automotive-Safety & Security 2017-Sicherheit und Zuverlässigkeit für automobile Informationstechnik. pp. 75–88. Gesellschaft für Informatik, Bonn (2017)
25. Lowe-Power, J., Ahmad, A.M., Akram, A., Alian, M., Amslinger, R., Andreozzi, M., Armejach, A., Asmussen, N., Beckmann, B., Bharadwaj, S., et al.: The gem5 simulator: Version 20.0+. arXiv preprint arXiv:2007.03152 (2020)
26. Maiza, C., Rihani, H., Rivas, J.M., Goossens, J., Altmeyer, S., Davis, R.I.: A survey of timing verification techniques for multi-core real-time systems. ACM Comput. Surv. 52(3) (jun 2019), `https://doi.org/10.1145/3323212`
27. Mascareñas González, A., Bouchebaba, Y., Santinelli, L.: Multicore shared memory interference analysis through hardware performance counters. In: 10thEuropean Congress on Embedded Real Time Software andSystems(ERTS 2020). Toulouse, France (Jan 2020), `https://hal.science/hal-02446031`
28. Mezger, B.W., Santos, D.A., Dilillo, L., Zeferino, C.A., Melo, D.R.: A survey of the risc-v architecture software support. IEEE Access 10, 51394–51411 (2022)
29. Moseley, T., Vachharajani, N., Jalby, W.: Hardware performance monitoring for the rest of us: a position and survey. In: IFIP International Conference on Network and Parallel Computing. pp. 293–312. Springer (2011)
30. Mushtaq, H., Al-Ars, Z., Bertels, K.: Survey of fault tolerance techniques for shared memory multicore/multiprocessor systems. In: 2011 IEEE 6th International Design and Test Workshop (IDT). pp. 12–17 (2011)
31. Nokovic, B., Sekerinski, E.: Model-based wcet analysis with invariants. Electronic Communications of the EASST 72 (Nov 2015), `https://eceasst.org/index.php/eceasst/article/view/2198`
32. Patel, A., Afram, F., Chen, S., Ghose, K.: Marss: A full system simulator for multicore x86 cpus. In: Proceedings of the 48th Design Automation Conference. pp. 1050–1055 (2011)
33. Piatka, C., Amslinger, R., Haas, F., Weis, S., Altmeyer, S., Ungerer, T.: Investigating transactional memory for high performance embedded systems. In: Architecture of Computing Systems – ARCS 2020: 33rd International Conference, Aachen, Germany, May 25–28, 2020, Proceedings. p. 97–108. Springer-Verlag, Berlin, Heidelberg (2020)
34. Sandberg, A.: Welcome and introduction to gem5 (2017), `https://www.youtube.com/watch?v=81lm0hp0t-M`
35. SiFive: Sifive tilelink specification. Tech. rep., SiFive (2023), `https://sifive.cdn.prismic.io/sifive/928d6a82-77a9-4291-8b60-5e815429b1ab_tilelink_spec_1.9.3.pdf`
36. Snyder, W.: Verilator 4.0: open simulation goes multithreaded. In: Open Source Digital Design Conference (ORConf) (2018)
37. Terpstra, W.: Tilelink: A free and open-source, high-performance scalable cache-coherent fabric designed for risc-v (2017), `https://www.youtube.com/watch?v=EVITxp-SEp4`, 7th RISC-V Workshop
38. Tousi, A., Iturbe, X.: Cortex-m-based soc design and prototyping using arm designstart. Tech. rep., Arm Limited (2018), `https://documentation-service.arm.com/static/5ed13515ca06a95ce53f9114?token=`
39. Waterman, A., Lee, Y., Avizienis, R., Patterson, D.A., Asanović, K.: The risc-v instruction set manual, volume ii: Privileged architecture, version 1.9. Tech. Rep. UCB/EECS-2016-129, EECS Department, University of California, Berkeley (July 2016), `https://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-129.pdf`
40. Waterman, A., Lee, Y., Patterson, D.A., Asanović, K.: The risc-v instruction set manual, volume i: User-level isa, version 2.1. Tech. Rep. UCB/EECS-2016-118, EECS Department, University of California, Berkeley (May 2016), `https://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-118.pdf`

41. Wilhelm, R., Engblom, J., Ermedahl, A., Holsti, N., Thesing, S., Whalley, D., Bernat, G., Ferdinand, C., Heckmann, R., Mitra, T., Mueller, F., Puaut, I., Puschner, P., Staschulat, J., Stenström, P.: The worst-case execution-time problem—overview of methods and survey of tools. ACM Trans. Embed. Comput. Syst. 7(3) (May 2008), `https://doi.org/10.1145/1347375.1347389`
42. Wu, L., Zhang, W.: A model checking based approach to bounding worst-case execution time for multicore processors. ACM Trans. Embed. Comput. Syst. 11(S2) (Aug 2012), `https://doi.org/10.1145/2331147.2331166`
43. Yao, G., Pellizzoni, R., Bak, S., Betti, E., Caccamo, M.: Memory-centric scheduling for multicore hard real-time systems. Real-Time Systems 48, 681–715 (2012)
44. Yoon, M.K., Kim, J.E., Sha, L.: Optimizing tunable wcet with shared resource allocation and arbitration in hard real-time multicore systems. In: 2011 IEEE 32nd Real-Time Systems Symposium. pp. 227–238. IEEE (2011)
45. Yourst, M.T.: Ptlsim: A cycle accurate full system x86-64 microarchitectural simulator. In: 2007 IEEE International Symposium on Performance Analysis of Systems & Software. pp. 23–34. IEEE (2007)
46. Yun, H., Yao, G., Pellizzoni, R., Caccamo, M., Sha, L.: Memguard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms. In: 2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS). pp. 55–64 (2013)

Axel Wiedemann received his bachelor's degree in 2019 and his master's degree in 2023, both in Computer Science in Engineering from the University of Augsburg, Germany. Since then he has been working in the Embedded Systems Group of Prof. Dr. Sebastian Altmeyer at the University of Augsburg. His research interests are in the area of system development for embedded applications with a focus on the use of FPGAs.

Florian Haas is a former postdoctoral researcher at the Chair for Embedded Systems at the University of Augsburg. His research interests are parallel embedded systems under real-time constraints with particular demands on performance and fault tolerance. He received his PhD degree in computer science from the University of Augsburg on fault tolerance of parallel applications on multi-core processors.

Sebastian Altmeyer began his academic career at the Compiler Design Lab at Saarland University, Germany, where he focused on timing verification of safety-critical real-time systems. After earning his PhD in 2012, he joined the Systems Engineering Group at the University of Amsterdam (UvA), broadening his research to design-space exploration, performance engineering, and computer architecture. In 2015, he moved to the Laboratory of Advanced Software Systems (LASSY) at the University of Luxembourg to work on cyber-physical systems design and modeling, before returning to University of Amsterdam where he was promoted to assistant professor. In 2019, Prof. Altmeyer was appointed full professor and head of the Embedded Systems Group at the University of Augsburg, Germany. Throughout his career, he has contributed to numerous national and international research projects, served in various program committees and earned several best and outstanding paper awards.

Received: July 24, 2024; Accepted: November 14, 2024.

Computer Science and Information Systems 22(2):623–648 https://doi.org/10.2298/CSIS240503020S

Resource-Aware Object Detection and Recognition Using Edge AI Across the Edge-Fog Computing Continuum*

Dragan Stojanović, Stefan Sentić, Natalija Stojanović and Teodora Stamenković

Faculty of Electronic Engineering, University of Niš
Aleksandra Medvedeva 14, 18000 Niš
dragan.stojanovic@elfak.ni.ac.rs
stefan.sentic@elfak.rs
natalija.stojanovic@elfak.ni.ac.rs
teodora.stamenkovic@elfak.rs

Abstract. Edge computing and edge intelligence have gained significant traction in recent years due to the proliferation of Internet of Things devices, the exponential growth of data generated at the network edge, and the demand for real-time and context-aware applications. Despite its promising potential, the application of artificial intelligence on the edge faces many challenges, such as edge computing resource constraints, heterogeneity of edge devices, scalability issues, security and privacy concerns, etc. The paper addresses the challenges of deploying deep neural networks for edge intelligence and traffic object detection and recognition on a video captured by edge device cameras. The primary aim is to analyze resource consumption and achieve resource-awareness, optimizing computational resources across diverse edge devices within the edge-fog computing continuum while maintaining high object detection and recognition accuracy. To accomplish this goal, a methodology is proposed and implemented that exploits the edge-to-fog paradigm to distribute the inference workload across multiple tiers of the distributed system architecture. The edge-fog related solutions are implemented and evaluated in several use cases on datasets encompassing real-world traffic scenarios and traffic objects' recognition problems, revealing the feasibility of deploying deep neural networks for object recognition on resource-constrained edge devices. The proposed edge-to-fog methodology demonstrates enhancements in recognition accuracy and resource utilization, validating the viability of both edge-only and edge-fog based approaches. Furthermore, experimental results demonstrate the system's adaptability to dynamic traffic scenarios, ensuring real-time recognition performance even in challenging environments.

Keywords: Resource awareness, Traffic Object Recognition, Edge Inteligence, Distributed Neural Networks, Edge-Fog Computing Continuum.

1. Introduction

Edge computing refers to the paradigm of processing data near its source or point of collection, rather than relying solely on centralized cloud servers. Edge intelligence involves the integration of artificial intelligence (AI) and machine learning (ML) algorithms into edge devices, enabling them to perform data analytics and decision-making tasks locally.

* The paper is an extension of the paper presented at RAW 2023 workshop

By bringing computation and intelligence closer to the data source, edge computing and edge intelligence offer numerous benefits, including reduced latency, improved bandwidth efficiency, enhanced privacy and security, and increased resilience to network failures. AI on edge devices and infrastructure powers real-time, context-aware, and intelligent applications across various fields, such as healthcare, smart cities, industrial automation, transportation, and agriculture. However, its potential comes with challenges, including limited resources, diverse device architectures, scalability difficulties, and concerns around security and privacy. The evolution of edge AI across the edge-fog computing continuum has been extensively explored in recent literature, highlighting the significance of distributed ML and AI in enabling real-time decision-making [21]. It underscores the importance of integrating ML and AI algorithms into edge devices and fog nodes to facilitate intelligent data processing, minimizing latency and network bandwidth usage [9].

In recent years, the rapid proliferation of Internet of Things (IoT) devices, including microcontrollers, single-board computers and smartphones, equipped with built-in or externally connected cameras, has led to their ubiquitous usage across a myriad of applications requiring detection and recognition of objects on the real-time video streams. This widespread adoption has heralded the advent of edge intelligent systems across diverse domains, with a notable emphasis on traffic object detection and recognition. Accurately identifying and classifying traffic objects, such as cars, trucks, motorcycles, bicycles, and pedestrians, is crucial for efficient traffic management, advanced driver assistance systems (ADAS), and autonomous driving. There is an increasing reliance on ML and deep learning (DL) algorithms for video stream analysis for object detection and classification in safety-critical embedded systems and IoT applications, such as autonomous driving systems, surveillance systems and security robots. It emphasizes the need for ML/DL technologies to meet strict timing requirements in real-time systems while maintaining accuracy, given the potentially catastrophic consequences of missed deadlines. Bian et al. in [3] aim to provide a comprehensive exploration of state-of-the-art results in ML/DL-based scheduling techniques, accuracy trade-offs, and security considerations in real-time IoT systems. The potential of deep neural networks (DNNs) to perform efficiently on edge IoT devices is particularly significant, as it harnesses the computational power and availability of these widely used devices. Exploring the intersection of advanced neural network architectures, real-time data processing, and distributed computing paradigms is essential for developing innovative solutions for traffic object detection, classification, and recognition [2]. By combining the proximity and processing capabilities of edge devices with fog servers and a cloud infrastructure, research efforts aim to enhance the efficiency, accuracy, and responsiveness of traffic object detection systems.

This paper addresses the challenges of training and deploying DNNs for traffic object detection and recognition across various edge devices, including Android smartphones, microcontrollers, and single-board computers. The focus is on distributing computational and inference tasks between IoT devices at the far edge, edge servers, and fog servers. We examine different deployment strategies, balancing trade-offs between model size, inference speed, power consumption, and accuracy. Specifically, we evaluate two approaches: (1) quantizing and optimizing the DNN model as TensorFlow Lite for direct deployment on edge devices, and (2) deploying the original DNN model on an edge or fog server. We also explore distributing the object recognition task by dividing it into two inference stages: object detection performed at the edge and object recognition of the detected items carried out on the fog server.

Through a series of experiments on traffic object detection and recognition, we evaluate the performance, accuracy, and resource consumption across different platforms, including microcontrollers, smartphones, single-board computers, and commodity servers, under various video data parameters and configurations. The results offer valuable insights into the practicality and efficiency of distributing DNNs for traffic object recognition from the edge to the fog. These findings support resource-aware edge intelligence by optimizing computational resource usage while preserving high recognition accuracy. Furthermore, this research lays the groundwork for developing intelligent transportation systems that harness the potential of edge devices, such as Android smartphones and microcontrollers, along with fog computing infrastructure, to improve traffic safety and management.

This paper represents the extended version of the paper presented at the RAW 2023 Workshop [20]. We now provide significant extension of the related work in object deetction and recognition at edge-fog infrastructure. Furthermore we give detailed explanation of the traffic object detection and recognition solutions implemented on an Android smartphone and a commodity PC, presented in the original paper. As the main improvement and the contribution of this paper we implemented and analysed new use cases for traffic object detection over novel edge devices using a TinyML approach. This involves deploying the traffic object detection solution on a microcontroller (Arduino Nano 33 BLE Sense) and a single-board computer (Raspberry Pi 4). Various DNN models suitable for object detection and recognition tasks have been utilized in their original versions and subsequently optimized, quantified and compressed to enable deployment on mentioned edge devices. Extensive experiments have been conducted to evaluate the performance and accuracy of these applications concerning ML tasks. The experiments also assessed resource usage, including memory and processing time. The results have been described and analysed in detail. The extended paper now provides thorough insights into the implementation and experimental evaluation of various traffic object detection and recognition tasks in different distributed configurations. It also covers the distribution of tasks across the edge-fog computing continuum, from microcontrollers and single-board computers to smartphones and fog servers (commodity PCs).

The paper is structured as follows. Section 2 presents research work in edge computing and edge intelligence related to object detection and recognition from video streams. Section 3 presents several strategies and corresponding applications for deployment of DNN for traffic object detection and recognition across various edge devices and a fog server. Section 4 presents the experimental evaluation for various traffic object detection scenarios and discusses the evaluation results. Section 5 gives concluding remarks and directions for future research.

2. Related Work

A growing body of research has focused on harnessing the power of edge devices, such as IoT devices and edge servers, to perform real-time object detection and recognition tasks at the network edge, minimizing latency and bandwidth consumption. The convergence of edge computing and DL methods and techniques has enabled the deployment of resource-efficient object detection and recognition models directly on edge devices, facilitating autonomous decision-making and edge AI applications.

Singh and Gill in [19] gives the extensive review of the unique characteristics and advantages of deploying AI algorithms directly on edge devices, enabling real-time inference and decision-making at the network edge. Furthermore, the survey discusses the diverse applications of Edge AI across domains such as smart cities, healthcare, autonomous vehicles, and industrial automation, highlighting its transformative potential in enhancing efficiency, scalability, and privacy in distributed computing environments. The article delves into the challenges and open research issues associated with Edge AI, such as resource constraints, security concerns, and algorithmic optimizations.

The need to integrate ML techniques into resource-constrained embedded devices, facilitated by advancements in technologies like the IoT and edge computing has given rise to TinyML, an embedded ML technique, a by enabling ML applications on low-cost, resource-constrained devices. However, implementing TinyML comes with challenges such as processing capacity optimization and maintaining model accuracy [10].

Shuvo et al. in [18] address the challenges of deploying DNNs on edge devices. It highlights the computational complexity and memory requirements of DNNs, which often necessitate cloud-based processing, leading to latency issues and security concerns. The paper explores optimization techniques at both hardware and software levels to enable efficient DNN deployment on edge devices, focusing on four research directions: novel DL architecture and algorithm design, optimization of existing DL methods, algorithm-hardware co-design, and efficient accelerator design. Through a comprehensive review, the paper provides insights into state-of-the-art tools and techniques for efficient edge inference, aiming to facilitate the integration of AI capabilities into next-generation edge devices.

The research on the distribution of DNNs for resource-aware systems has gained significant attention in recent years. Several studies have explored different approaches and strategies for optimizing the training and deployment of DNNs in various domains. In the context of object detection and recognition from edge to cloud, several relevant research papers provide valuable insights and inspiration. Bittencourt et al. in [4] discuss the integration and challenges of the IoT, fog, and cloud continuum, highlighting the need for efficient resource utilization. Lockhart et al. in [13] propose Scission, a performance-driven and context-aware cloud-edge distribution approach for DNNs, emphasizing the importance of considering context and performance in distribution decisions. Cho et al. in [5] present a study on DNN model deployment on distributed edges, focusing on distributed inference across edge devices.

Lin et al. in [12] propose a distributed DNN deployment approach from the edge to the cloud for smart devices, addressing the challenges of efficient utilization of resources in different computing tiers. McNamee et al. in [15] advocate for adaptive DNNs in edge computing, emphasizing the need for dynamic adaptation to optimize resource usage. Ren et al. in [17] provide a survey on collaborative DNN inference for edge intelligence, exploring the collaborative aspects of inference across edge devices. Hanhirova et al. in [7] characterize the latency and throughput of convolutional neural networks (CNN) for mobile computer vision, providing insights into the perfor-mance aspects of DNNs on resource-constrained devices.

Lee et al. in [11] propose Transprecise Object Detection (TOD) for maximizing real-time accuracy on the edge, highlighting the importance of accurate object detection for edge scenarios. Parthasarathy et al. in [16]introduce DEFER, a distributed edge inference approach for DNNs, focusing on resource-efficient inference in distributed edge environments. Teerapittayanon et al. in [22] investigate distributed DNNs over the cloud, edge, and end devices, highlighting the trade-offs between resource utilization and computational capabilities across different components of the system architecture.

In line with our research, Dharani et al. in [6] present the utilisation of TinyML and TensorFlow Lite on mobile phones for image classification, but without more extensive experimental evaluation and discussions. Akhtar et al. in [1] introduce multiple real-time deployable cost-efficient solutions for motorbike detection using state-of-the-art embedded edge devices, addressing the critical need for accurate and real-time traffic surveillance and road safety. The paper presents an improved baseline accuracy of motorbike detection by developing a custom network based on YOLOv5, the part of the You Only Look Once (YOLO) family.

The aforementioned papers contribute to the understanding of resource-aware DNN deployment and optimization techniques in various contexts. Our research aims to provide insights into the efficient training and deployment of DNNs for traffic object detection, classification and recognition, considering the resource utilization from edge to fog in the context of edge devices with various computing capabilities and resources available.

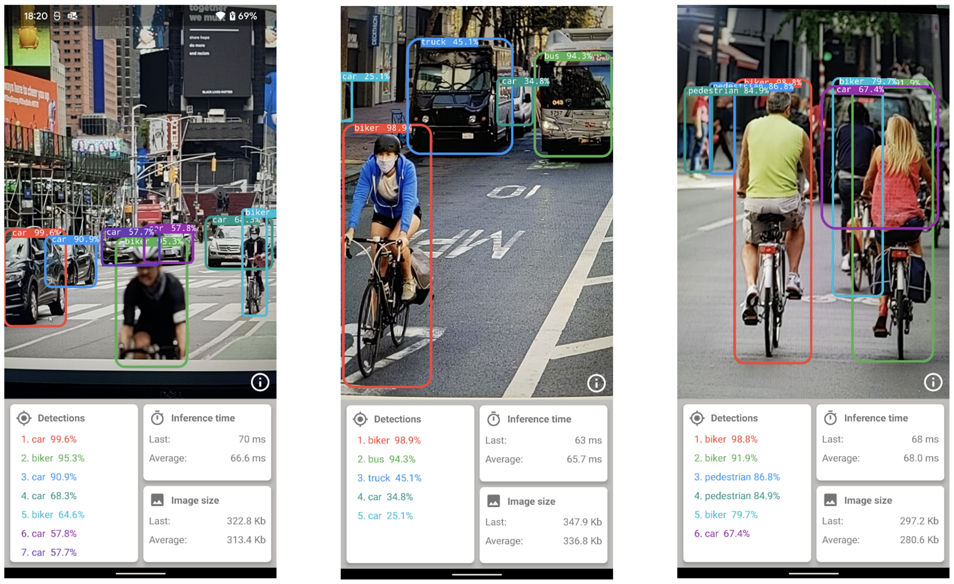
3. Traffic Object Detection and Recognition Across the Edge-Fog Continuum

DNNs have shown promising results in various computer vision tasks, including object detection and recognition. In this Section we present three case studies related to distribution of DNN-based software components in the context of traffic object detection and recognition.

3.1. Traffic Object Recognition

To recognize traffic objects captured by a camera on Android smartphones, we implement two approaches: the first executes entirely on the edge device, while the second divides the process between the edge device and a fog node. The training of the DNN model is based on the TensorFlow Object Detection API and specifically utilizes the MobileNetV2 SSD 320x320 coco17 tpu-8 pre-trained model. To enhance the training process, we have utilized video data captured from an Android phone camera, as well as publicly available traffic video datasets such as the Udacity Self Driving Car Dataset [23], INRIA Graz-02 (IG02)[14], and the Bike-rider Detector dataset [24]. Manual labeling was applied to these datasets when necessary to ensure accurate annotation of traffic objects, for detection of cars, trucks, motorcycles, bicycles, and pedestrians.

The first approach we propose utilizes the computing power and resources available solely on the smartphone, making it an offline approach. This method does not require a network connection for traffic object recognition within the Android application. To accommodate the limited resources available on the smartphone, the TensorFlow model used for recognition is quantized and converted to TensorFlow Lite form. By optimizing the model, we ensure that it can efficiently operate on the smartphone without compromising its performance. The trained model is integrated into the Android application, enabling real-time traffic object recognition directly on the smartphone without the need for an internet connection (Figure1). Figure 2 illustrates the execution flow of object recognition



Detections: 1. car 99.6% 2. biker 95.3% 3. car 90.9% 4. car 68.3% 5. biker 64.6% 6. car 57.8% 7. car 57.7%
Inference time: Last: 70 ms Average: 66.6 ms
Image size: Last: 322.8 Kb Average: 313.4 Kb

Detections: 1. biker 98.9% 2. bus 94.3% 3. truck 45.1% 4. car 34.8% 5. car 25.1%
Inference time: Last: 63 ms Average: 65.7 ms
Image size: Last: 347.9 Kb Average: 336.8 Kb

Detections: 1. biker 98.8% 2. biker 91.9% 3. pedestrian 86.8% 4. pedestrian 84.9% 5. biker 79.7% 6. car 67.4%
Inference time: Last: 68 ms Average: 68.0 ms
Image size: Last: 297.2 Kb Average: 280.6 Kb

Fig. 1. Android application for traffic object recognition

conducted within an Android application. Once the camera image is available, it is sent to the detector component. In offline mode, the *LocalDetector* component is employed, utilizing a TensorFlow Lite model deployed on the Android smartphone. Before inputting the image into the model, specific preparations must be completed, including scaling and rotation. Furthermore, after detection, it is crucial to parse the results and generate objects that will be used for GUI creation.

The second approach utilizes the advantages of fog computing by offloading part of the processing to a fog server. In this method, the video captured by the smartphone camera undergoes preprocessing within the Android application. These preprocessing steps may include scaling, rotation, and filtering of the captured images to enhance the quality and clarity of the input data. The preprocessed images are then encoded in Base64 format and sent to the fog server through a Web socket using the SocketIO library. The fog server, implemented with Flask/Python, hosts the original TensorFlow model, which performs object recognition on the received images. The results of this recognition process are returned to the Android application in JSON format, providing real-time feedback and visualization of the recognized traffic objects.

The execution flow of object recognition in edge-fog scenario is illustrated in Figure 3. The client application captures an image from the camera and processes it before sending it to the server. This processing includes scaling, rotation (considering the device's

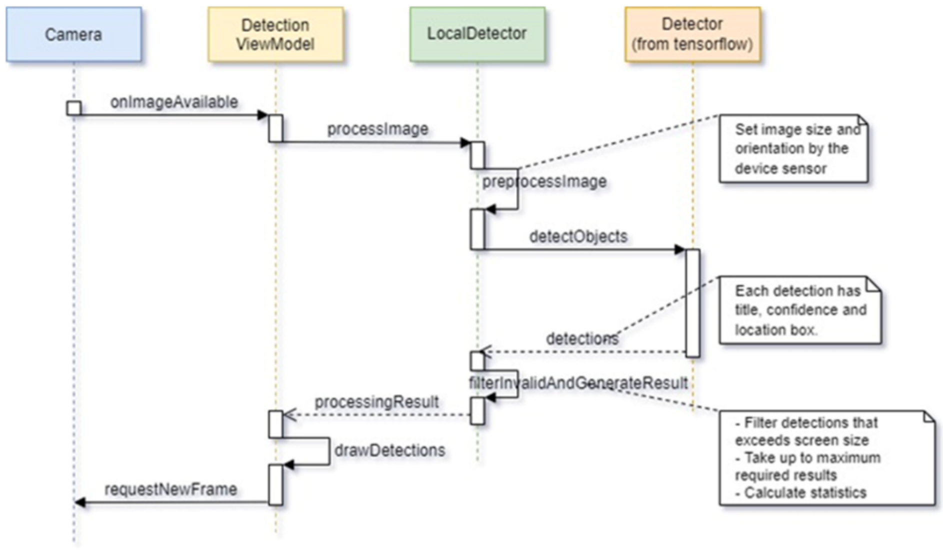

Camera
Detection ViewModel
LocalDetector
Detector (from tensorflow)
onImageAvailable
processImage
preprocessImage
Set image size and orientation by the device sensor
detectObjects
Each detection has title, confidence and location box.
detections
filterInvalidAndGenerateResult
processingResult
- Filter detections that exceeds screen size
- Take up to maximum required results
- Calculate statistics
drawDetections
requestNewFrame

Fig. 2. The sequence diagram during the detection process on Android smartphone.

sensor), and encoding the image into Base64 format (as a string). The image is transmitted to the server as an emitted event indicating that it is ready for processing. Upon receipt, the server decodes the Base64 string to reconstruct the image. On the server side, the TensorFlow library is utilized to run the model and perform object detection within the frame. The identified objects are then returned to the client in JSON format.

The second method distributes the computational workload between the smartphone and the fog server, offloading resource-intensive tasks to a more powerful computing infrastructure. This approach enhances the accuracy and robustness of traffic object recognition, particularly in situations where the smartphone's resources are limited. Additionally, leveraging fog computing alleviates the strain on the smartphone's battery and processing capabilities, resulting in better performance and an improved user experience. Both methods provide unique advantages regarding resource utilization, real-time performance, and accuracy, addressing various requirements and constraints, which are experimentally evaluated and discussed in the following section. The source code for the Android application that implements traffic object recognition using both methods is available on GitHub [1].

3.2. Car Model Recognition

The second use case is related to car model recognition. To address the specific challenge of car model recognition, we propose a two-stage approach that leverages both edge and fog computing resources. Our method is built on the pre-trained MobileNetV2 model, known for its outstanding performance across various computer vision tasks. For training the model specifically for car model recognition, we employed the Stanford Cars dataset, which contains more than 16,000 images and includes more than 190 different car classes.

[1] https://github.com/drstojanovic/camera

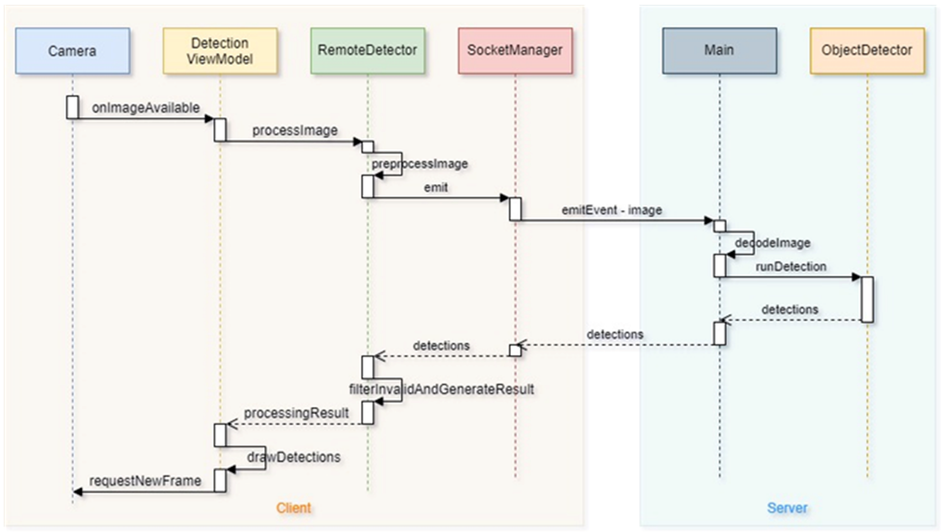


Fig. 3. Sequence diagram for the server-based object detection

The execution flow for recognizing car models is illustrated in Figure 4. As depicted, the process occurs in multiple phases, with some tasks handled on the edge side (client) and others on the fog side (server).

Since an existing object detector is used, it will return detections for all five classes of traffic objects: cars, trucks, motorcycles, bikes, and pedestrians. The first step is to filter out only the class of interest, cars. After capturing the image from the camera, the following steps are taken:

1. Object detection is conducted using an edge model, which discards all objects not belonging to the ”car” class.
2. Based on the detections, the image is cropped, creating a list of car crops.
3. Each crop is then encoded in Base64 format, and this list of strings is sent to the server.
4. Upon receipt, the server decodes the list back into images. Each crop is resized to fit the model’s input size (192x192 in this application).
5. The model is then executed on each crop, providing a list of potential classes for each image.
6. The results are formatted as JSON, resulting in a list of lists whose length corresponds to the number of detected cars or crops obtained.
7. On the edge side, the detection results are merged with the classification results from the server, and a bounding box is drawn around each car, displaying the recognized class.

The high-level flow of the car model recognition is illustrated in Figure 5. In the first stage of our approach, the Android application takes advantage of edge computing capabilities to detect cars and determine their positions within the video images. We employ a TensorFlow Lite model, like the one used in the previous implementation, to perform

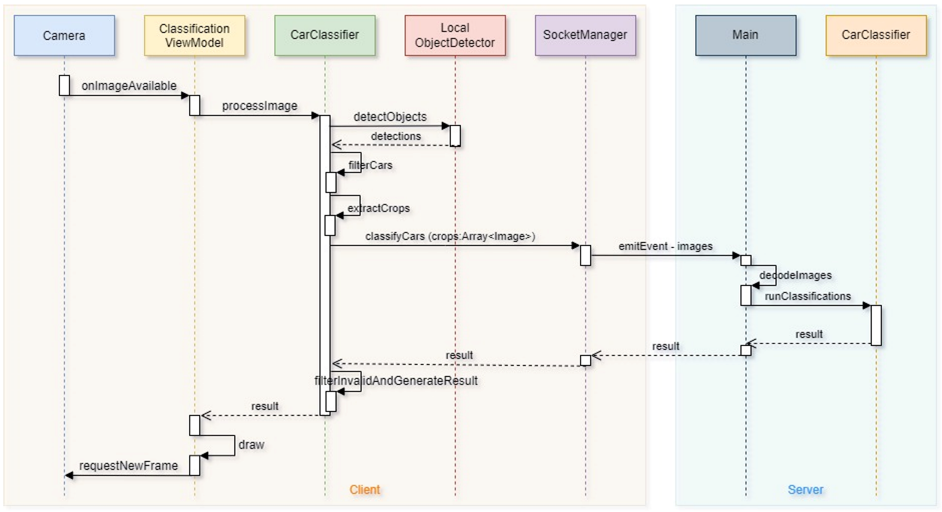


Fig. 4. Execution flow of a car model recognition

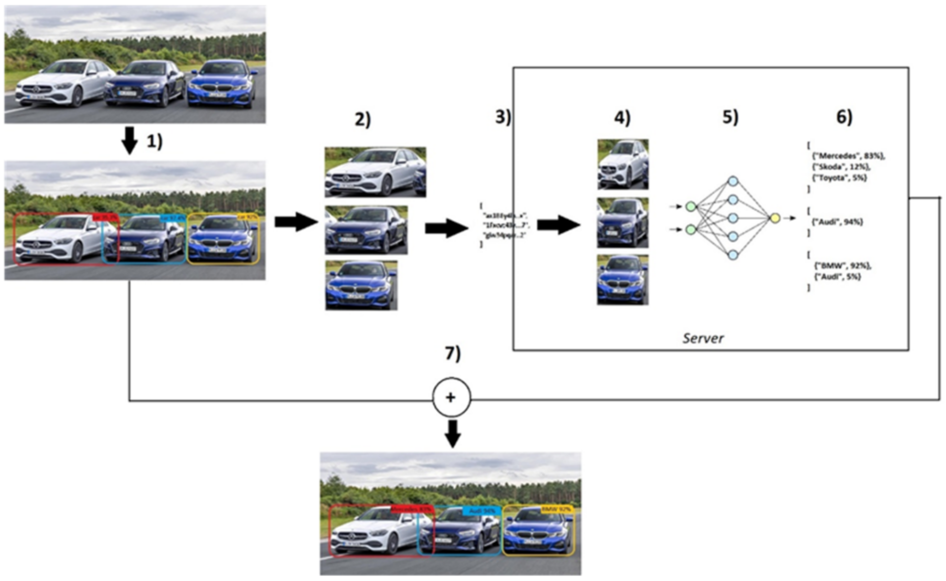
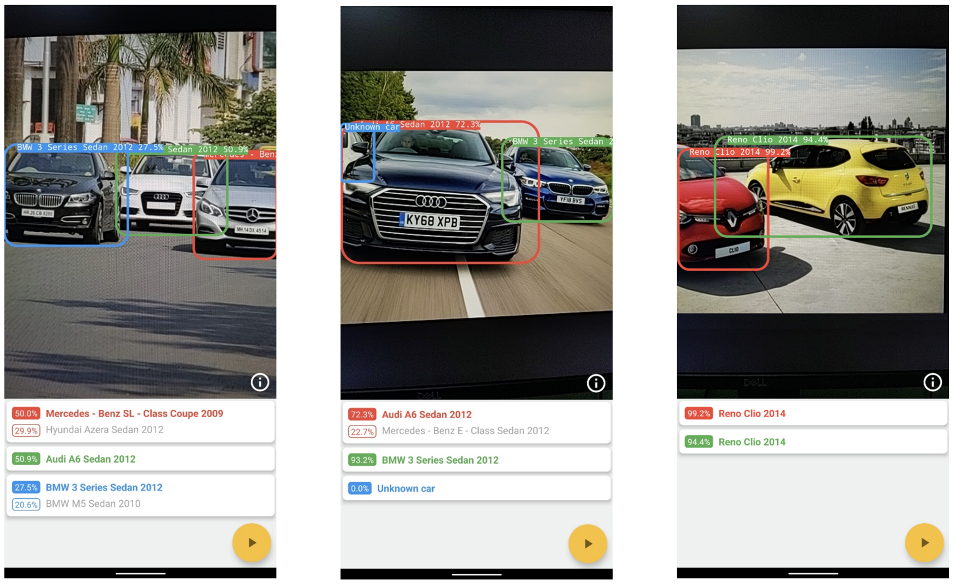

1)
2)
3)
4)
5)
6)
7)
Server
[("Mercedes", 83%), ("Skoda", 12%), ("Toyota", 5%)]
[("Audi", 94%)]
[("BMW", 92%), ("Audi", 5%)]

Fig. 5. The steps performed for car model detection.

this initial car detection task on the smartphone. Once the cars are identified, the corresponding regions of interest (ROIs) are extracted from the video frames and preprocessed to enhance their quality and suitability for subsequent recognition.

The preprocessed and cropped car images are then sent from the Android application to the fog server. The fog server application handles the second stage of the car model recognition process. It employs a trained TensorFlow model, specifically designed for the Stanford Cars Dataset, to identify the model of each car in the received images and sends the results back to the Android application (Figure6). The fog server's superior computational resources and processing power allow it to perform more intensive tasks, such as detailed car-type classification. To enable communication between the Android appli-



50.0% Mercedes - Benz SL - Class Coupe 2009
29.9% Hyundai Azera Sedan 2012

50.9% Audi A6 Sedan 2012

27.5% BMW 3 Series Sedan 2012
20.6% BMW M5 Sedan 2010

72.3% Audi A6 Sedan 2012
22.7% Mercedes - Benz E - Class Sedan 2012

93.2% BMW 3 Series Sedan 2012

0.0% Unknown car

99.2% Reno Clio 2014

94.4% Reno Clio 2014

Fig. 6. Car model detection in the Android application

cation and the fog server, we use WebSockets. This approach facilitates real-time, bidirectional communication, allowing for the efficient transfer of preprocessed car images from the smartphone to the server and the return of recognition results to the mobile application. By utilizing WebSockets, we ensure a seamless and responsive user experience throughout the car model recognition process.

In the training phase, the Table 1 shows hyperparameter settings that were used for MobileNetV2 and MobileNetV2 SSD algorithms.

By leveraging both edge and fog computing resources, our two-stage approach optimizes the distribution of computational tasks. The edge computing performed on the smartphone efficiently detects cars and extracts relevant regions of interest (ROIs), which reduces the volume of data sent to the fog server. This strategy minimizes bandwidth usage and latency. The more resource-intensive task of car model recognition is offloaded to the fog server, taking advantage of its greater computational capabilities and trained

Table 1. Hyperparameter settings

Epochs	100
Learning rate	0.1
Quantization	fp16
Training:Validation (%)	80:20
Batch size	64
Momentum	0.9
Weight decay	0.001

model. This approach maximizes the utilization of computing resources and enhances the overall performance and accuracy of car model recognition in our system. The source code for traffic object recognition server is available at GitHub[2] while its docker image can be pulled from Docker Hub[3].

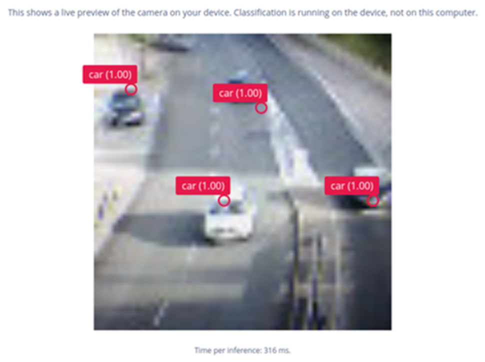
3.3. Vehicle detection on microcontrollers and single-board computers

The third use case focuses on vehicle detection on low-resource edge devices, such as microcontrollers and single-board computers. The goal is to detect and recognize cars in video streams captured by integrated cameras. This involves implementing various object detection model architectures and optimizing them for execution on devices like the Arduino Nano 33 BLE Sense (Arduino Nano) and Raspberry Pi 4 Model B (RPi). Part of the model training and evaluation was conducted using the Edge Impulse platform [8], an online MLOps platform that supports the training, testing, and deployment of ML/DL models across a wide range of edge devices. Additionally, transfer learning was applied using TensorFlow Lite Model Maker. The training dataset, obtained from Kaggle, comprises 499 images containing a total of 4,281 vehicles, with each frame featuring between 4 and 13 vehicles. In addition to collecting data directly from the edge devices, datasets were also uploaded to the Edge Impulse platform in YOLO txt format. In this format, each image has an associated .txt file listing the detected objects, where each line in the file represents a single object, containing its class and the normalized coordinates of its bounding box.

The initial implementation of vehicle detection was performed on the Arduino Nano 33 BLE Sense, a resource-constrained edge device. Given the limited capabilities of this development board, the trained models operate on smaller image dimensions. However, increasing image size leads to longer inference times and larger model sizes. Unfortunately, due to the dataset containing small objects, models trained on low-resolution images yielded poor results. For the Arduino Nano application, two models, FOMO MobileNetV2 0.1 and FOMO MobileNetV2 0.35 were trained. The values 0.1 and 0.35 represent the alpha parameters in the MobileNetV2 architecture, indicating the network's width by scaling the number of channels in each layer. These hyperparameters help balance model size, computational efficiency, and accuracy. FOMO models were trained for various numbers of epochs using both RGB and Grayscale images, with various hyperparameter configurations, to find the optimal trade-off between performance and accuracy.

[2] https://github.com/drstojanovic/trafficAssistantServer
[3] https://hub.docker.com/r/stefan2708/ta_server

The Edge Impulse platform was used to generate binary files for model inference on the Arduino device. Inference can be initiated with the command *edge-impulse-run-impulse*. Object detection results from the OV7675 camera attached to the Arduino Nano are displayed in the browser, as shown in Figure 7.


This shows a live preview of the camera on your device. Classification is running on the device, not on this computer.

Time per inference: 316 ms.

Fig. 7. Object detection from Arduino camera shown in browser

The Raspberry Pi 4 Model B offers greater computing resources, enabling the training of a broader range of models. Beyond the FOMO algorithms, the following models were also trained using the Edge Impulse platform:

1. MobileNetV2 SSD FPN-Lite: This model was pre-trained on the COCO 2017 dataset with images of size 320x320. It consists of three parts:
   - Basic network (MobileNetV2): Provides high-level features for classification or detection. By removing the fully connected and softmax layers and adding a detection network, the model can determine object locations in the image.
   - Detection network (Single Shot Detector, SSD): Detects multiple objects in an image using a single CNN SSD models are faster and more efficient as they simultaneously predict object classes and regions containing objects.
   - Feature Pyramid Network (FPN): Utilizes an input image of a single size to generate feature maps for different sizes, facilitating the detection of objects of various sizes.
2. YOLOv5: This model performs object detection in a single pass. Introduced in 2020, YOLOv5 incorporates the EfficientDet architecture, built on EfficientNet, to optimize both resource usage and accuracy. Unlike its predecessor, YOLOv5 abandons anchor-based detection, instead relying on a convolutional layer to predict bounding box coordinates directly.

YOLOv5 is used in the transfer learning process, leveraging prior training on a larger dataset. This enables the model to learn from a broader data set and improve its generalization capabilities. During the training of the YOLOv5 model on Edge Impulse, the model size can be selected: Nano, Small, Medium, and Large . Due to resource limitations, the Nano model with 1.9M parameters, sized at 3.78 MB, was chosen. Models from

Edge Impulse are downloaded in the EIM (Edge Impulse Model) format. EIM files are binary files for Linux and macOS that encapsulate the complete impulse built on the Edge Impulse platform, including signal processing, model, and inference blocks. EIM files are architecture-specific and allow direct inference execution on the RPi device. The hyperparameters defined for training four mentioned models that have been deployed and run at Arduino Nano and RPi are given in Table 2.

Table 2. The hyperparameters defined for training NN models

Hyperparameters	FOMO 0.1	FOMO 0.35	MobileNetV2 SSD FPN	YOLOv5 (Nano)
Epochs	60	60	25	30
Learning rate	0.001	0.001	0.15-0.01	0.01-0.001
Quantization	int8	int8	int8	fp16
Training:Validation (%)	80:20	80:20	80:20	80:20
Batch size	32	32	32	16-32
Momentum	0.9	0.9	0.95	0.937

Inference on the RPi device is performed on images generated using a camera connected to the Arduino development board. Frames are captured on the Arduino and transmitted to the RPi device using serial communication. The OV7675 camera records images sized at 320x240 in RGB565 format. Within the Python script on the RPi device, bytes are read from the serial port, and conversion from RGB565 to RGB888 format is carried out. The converted image is passed to the `run_inference` function, which performs preprocessing and inference (Figure 8). TensorFlow Lite Model Maker [4] is a library designed

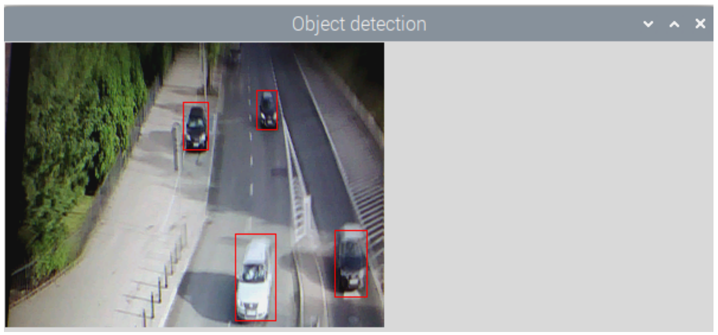


Fig. 8. Detection of vehicles using DL models deployed on RPi

for training TensorFlowLite models with custom datasets. It leverages transfer learning to minimize the amount of training data required and reduce training time. For object detection, the library offers five versions of EfficientDet-Lite models, each differing in memory usage, detection latency, and mean Average Precision (mAP). These models are deployed

[4] https://www.tensorflow.org/lite/models/modify/model_maker

on the RPi for real-time vehicle detection. In this setup, frames transmitted from the Arduino Nano's camera are used for detection. However, the inference process differs here, as it utilizes the TensorFlowLite Interpreter, which requires a tensor as input. This necessitates preprocessing the image and generating a tensor with the correct shape and data type. Finally, the function completes the process by drawing bounding boxes and saving the annotated image. The source code of the Arduino Nano and RPi applications are available at GitHub [5].

4. Resource-Aware Experimental Evaluation

This section presents an experimental evaluation of previously described use cases and the corresponding methods for distributing DNNs in traffic object detection and recognition. In the first use case, we assess two solutions: one executed entirely within the Android application using a TensorFlow Lite model, and the other performed on the fog server. In the third use case, we evaluate solutions implemented on Arduino Nano and RPi devices. To measure performance, accuracy, and resource consumption (CPU, memory, and energy), we conduct experiments using various video data parameters and configurations. The objective is to explore the trade-offs between different approaches across the edge-fog computing continuum and analyze their behavior under varying conditions. For evaluation, we use representative datasets that include a variety of traffic scenarios and object types, ensuring coverage of diverse lighting conditions, weather patterns, and traffic densities.

4.1. Smartphone and Server implementation

To evaluate the Android application solution, we measure its performance and accuracy directly on the smartphone. The computational requirements, including CPU usage, memory consumption, and energy consumption, are analyzed using the Android Profiler tool. Furthermore, we assess the accuracy of traffic object recognition by comparing the application's outputs with ground truth annotations from the datasets.

Similarly, for the fog server solution, we assess its performance, accuracy, and resource consumption. The server's computational requirements, such as CPU usage, memory utilization, and energy consumption, are analyzed. Additionally, the recognition results from the fog server are compared with ground truth annotations to evaluate the solution's accuracy. A timeline diagram illustrating CPU and memory consumption, along with energy usage for traffic object recognition performed on the Android smartphone and the fog server, is presented in Figure 9. The maximum CPU utilization during local detection on the Android smartphone reached 33%, while server-based detection peaked at 14%. The maximum RAM usage during local detection was 316 MB, compared to 240 MB for server-based detection. The TensorFlow model size is 11.2 MB; however, with model quantization, it can be reduced to 3.2 MB.

During the experimental evaluation, we varied the video data parameters and configurations to analyze the performance of both solutions under different conditions. This involved adjusting factors such as video resolution, frame rate, lighting conditions, and

[5] https://github.com/drstojanovic/object-detection-rpi

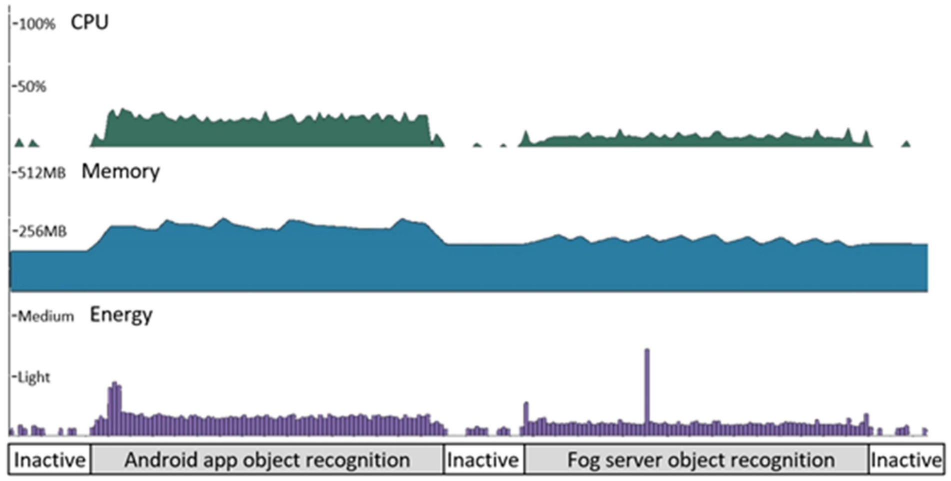


Fig. 9. CPU, memory, and energy usage for edge (smartphone) and edge server solutions

traffic densities. By conducting experiments across a range of scenarios, we aim to provide a thorough assessment of each solution's performance and resource utilization. The experimental evaluation is carried out using appropriate benchmarking tools and metrics to ensure reliable and meaningful results. We measure the execution time, resource utilization, and accuracy of both solutions across various datasets and configurations. The collected data is analyzed and compared to identify the strengths and weaknesses of each approach. For all experiments, a Google Pixel 4 phone was used, while the server application ran on a Lenovo Legion laptop (CPU: i5-9300H, RAM: 16.0 GB, GPU: NVIDIA GeForce GTX 1650). The server is configured on a local network to facilitate access by the mobile application.

The first experiment aimed to evaluate the recognition accuracy across different image resolutions and object sizes, with the latter expressed as a percentage relative to the overall image size. The phone's camera was pointed at a computer monitor displaying an image of a car that progressively decreased in size, simulating the appearance of traffic objects at varying distances. Testing was conducted for both operational modes at each of the four available resolutions in the application settings. The image quality was set to the maximum (100 %, with no compression). The reliability of detection, expressed as percentages, is presented in Table 3. This experiment showed that the reliability values for detection, and thus the overall detection quality, are quite comparable for both edge detection and server-based detection. A slight advantage was noted for edge detection at lower image resolutions, likely because of the extra image processing involved in sending the image to the server (such as encoding and decoding in Base64 format). One potential solution to improve performance is to use a more advanced image transport method, like implementing one of the transfer protocols specifically designed for image handling. We also assessed how performance (execution speed) depends on image resolution and image quality, with values expressed in milliseconds (ms). This experiment aimed to evaluate the impact of image compression on detection speed and image size. The phone's camera was pointed at a computer monitor displaying a consistent image of a car. Testing was con-

Table 3. Dependency of detection accuracy on image resolution and object size

Object size	Resolution							
	640 x 640		512 x 512		300 x 300		160 x 160	
	Edge	Fog	Edge	Fog	Edge	Fog	Edge	Fog
100%	98.9	97.8	98.7	98.7	98.9	98.3	90	84.3
80%	95.4	95	95.1	95.4	94.7	93.1	82.4	82.1
50%	89.2	90	87.4	88.7	89.9	87.9	83.4	80.5
30%	47.1	42.3	40.1	32.8	31.7	19.7	23.1	17.4
15%	12.3	12.7	10.7	12.1	10.2	2.1	7.4	1.8

ducted for both operating modes across all four available resolutions, using image quality levels of 100 %, 70 %, 50 %, 30 %, and 20 % achieved via JPEG compression.

Table 4 illustrates how detection speed varies with changes in image characteristics. The rows represent a decrease in image quality, while the columns show a reduction in resolution. The values are presented in milliseconds, and it is evident that local detection operates at a significantly higher speed compared to the server-based detection.

Table 4. Dependency of performance (speed of execution) on image resolution and image quality (in ms)

Quality	Resolution							
	640 x 640		512 x 512		300 x 300		160 x 160	
	Edge	Fog	Edge	Fog	Edge	Fog	Edge	Fog
100%	61.4	153.2	58	137	55.9	103.2	45.8	83.9
70%	56.5	112.2	50.4	99.8	43.9	87.7	43	76.9
50%	55.5	108.2	49.8	98.9	43.6	86.7	42.9	76.4
30%	55.3	103.8	49.1	93.1	43.4	84.7	43.4	75
20%	54.4	101.8	48.9	90.2	43.9	84.1	42.8	74.9

For local detection, the difference in detection speed between a resolution of 640x640 resolution and 100 % image quality and a resolution of 160x160 and 20 % quality is approximately 18.6 milliseconds per frame While precise measurement is challenging, according to results presented in Tables 3 and 4 it is evident that the detection quality for smaller objects is significantly reduced at lower resolution and image quality settings.

When it comes to server-based detection, internet connection speed becomes a significant factor. Since a GPU was utilized, the model execution itself was short (45ms), with most of the time being spent on image transportation. During the testing, an internet speed of 55.8 Mbps for download and 7.7 Mbps for upload was used. In the case of edge detection, the byte size of the image does not have as much influence as it does for server-based detection, as each image is transported to the server, and any reduction in image size contributes to increased detection speed. The difference in detection speed between settings A (640x640 resolution and 100 % quality) and settings B (160x160 resolution and 20 % quality) amounts to 78.3 milliseconds per frame. A significant degradation in detection quality on the server compared to edge detection was observed when decreasing the image resolution.

By assessing the performance, accuracy, and resource consumption of both the Android application and the fog server solutions, we aim to shed light on the trade-offs associated with each approach. This experimental evaluation will enhance our understanding of how these distribution methods perform in traffic object recognition tasks. Ultimately, this analysis will aid in choosing the most suitable solution based on specific requirements, including resource availability, real-time performance, and accuracy.

4.2. Arduino Nano and RPi Solutions for Object Detection

Testing and evaluating vehicle detection models involved comparing key parameters essential for implementing AI at the edge. Resource utilization metrics, such as RAM and flash memory usage, are particularly important for edge devices. Additionally, the time required for object detection is critical for real-time decision-making scenarios. A comparison of model accuracy was also conducted. The objective is to compare models developed using the Edge Impulse platform and TensorFlow Lite tools, exploring various combinations of preprocessing and hyperparameters. Furthermore, this comparison encompasses devices with differing resources, specifically the Arduino Nano and RPi.

The comparison between the FOMO 0.1 and FOMO 0.35 models on the Arduino Nano has been conducted. All results are presented for the quantized versions of the models, utilizing integer 8-bit values. The EON model format was chosen due to its lower resource overhead compared to TFLite models. Figure 10 illustrates the comparison of these models across different image sizes, focusing on model accuracy and inference time. The diagrams indicate that the accuracy of the FOMO 0.1 and FOMO 0.35 models is comparable across all image sizes. However, FOMO 0.1 shows significantly better performance for smaller dimensions, such as 64x64, whereas the advantage shifts toward the FOMO 0.35 algorithm for larger dimensions. Inference time increases with image size, reaching 3 seconds for dimensions of 160x160. Furthermore, the inference time of the FOMO 0.35 model diverges considerably from that of the FOMO 0.1 model as the dimensions increase.

Figure 11 illustrates the memory utilization for the same models. Although flash memory usage remains consistent across various image dimensions, maximum RAM utilization rises sharply as the dimensions increase. The diagram also shows the available RAM on the Arduino development board; models that exceed this limit cannot be executed on the device.

The evaluation of the models executed on the Raspberry Pi includes those generated on Edge Impulse as well as models created using TensorFlow Model Maker. Figure 12 illustrates the changes in accuracy and inference time for different models with varying image sizes. The YOLOv5 model demonstrates superior accuracy in all scenarios, except for the smallest image size. However, in terms of object detection speed, the YOLOv5 algorithm requires up to seven times longer to make decisions compared to the FOMO algorithms.

The resource utilization diagram (Figure 13) reveals that the maximum RAM usage with FOMO algorithms exceeds that of the YOLO algorithm at higher dimensions. Flash memory utilization remains consistent across all dimensions. Notably, the FOMO algorithms require approximately 70 KB, while the YOLOv5 model occupies 1.8 MB. On

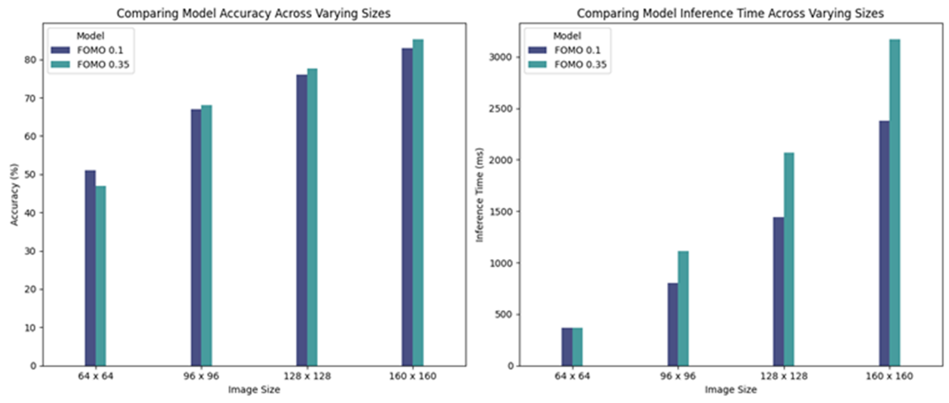


Fig. 10. Comparison of NN models based on accuracy and inference time.

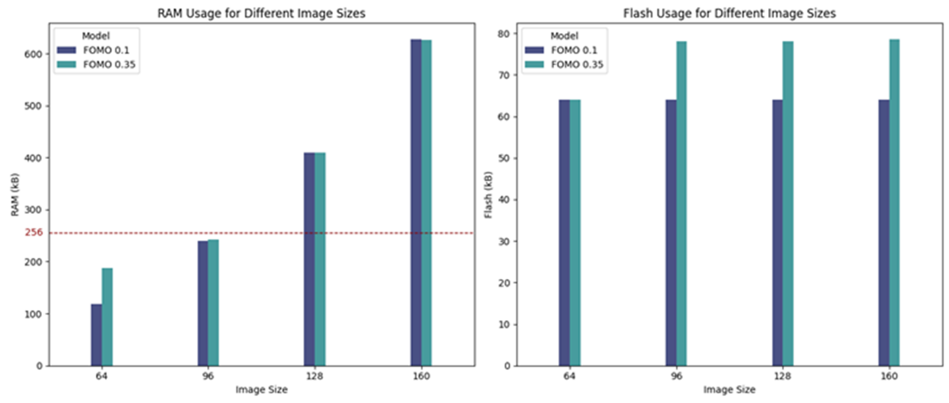


Fig. 11. Comparison of NN models based on memory utilization

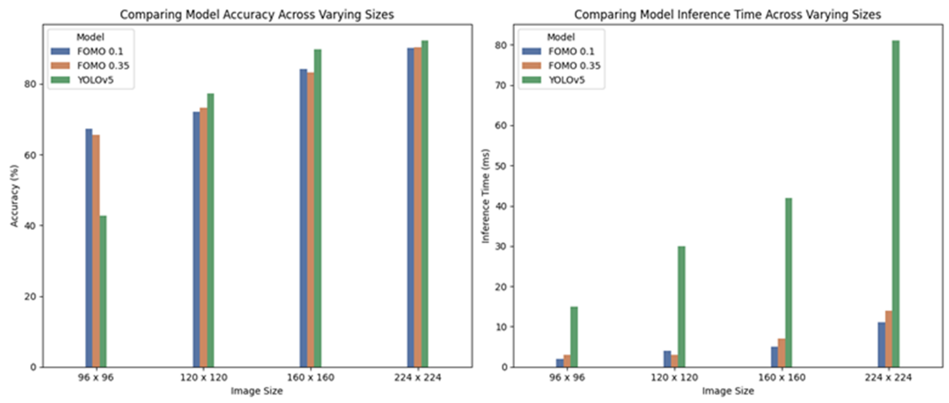


Fig. 12. Comparison of NN models based on accuracy and inference time on RPi

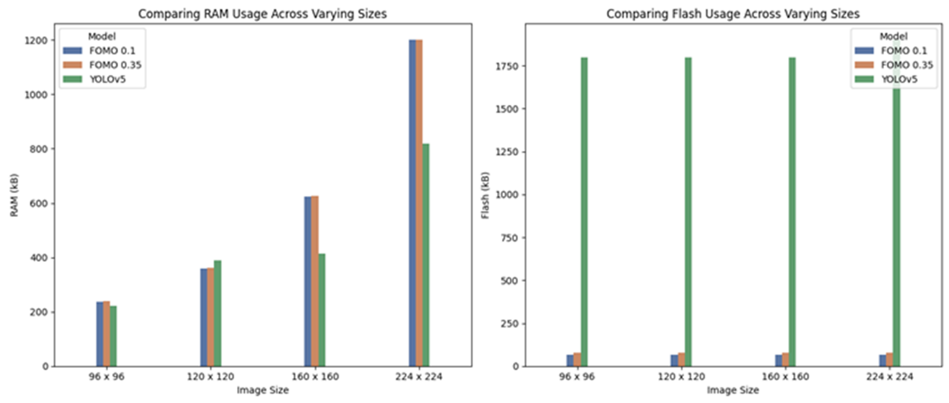


Fig. 13. Comparison of NN models based on memory utilization on RPi

Edge Impulse, several model optimizations are available. Figure 14 illustrates the differences between EON models with float32 values and their quantized versions using int8 values. Notably, quantization has a substantial effect on the size of the MobileNetV2 SSD and YOLOv5 models, as well as on inference time. The numerical comparison of the

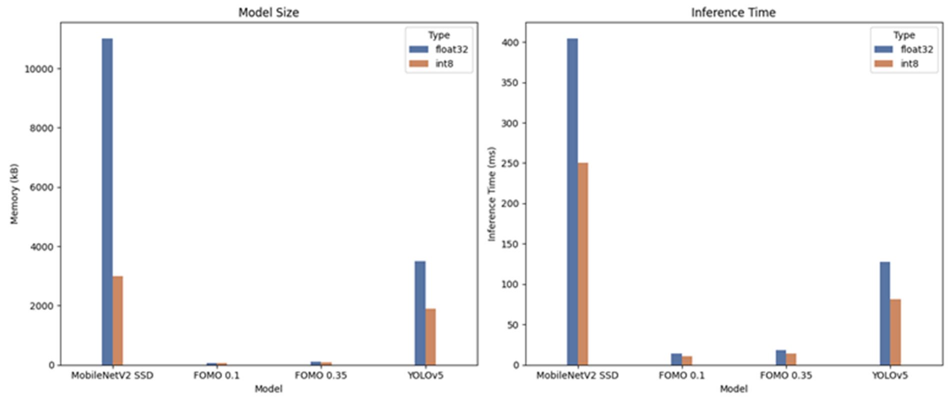


Fig. 14. Comparison between float32 and int8 NN model versions

models' size and inference time values is given in Table 5. The comparison of models compiled using EON and TensorFlow Lite compilers is shown in Figure 15, and presented in Table 6). The objective of the EON compiler is to reduce resource overhead, while maintaining unchanged accuracy and inference time. The values are presented for the int8 versions of the models. Resource optimization using the EON compiler is not supported for YOLOv5.

TensorFlow Model Maker enables training of five versions of EfficientDet models (1-5). The results of the trained EfficientDet2 model are presented in Table 7. Due to the

Table 5. The NN model size and inference time values

	Model size (float32)	Model size (int8)	Inference time (float32)	Inference time (int8)
MobileNetV2 SSD	11 MB	3 MB	404 ms	249 ms
FOMO 0.1	66.9 KB	64.4 KB	14 ms	11 ms
FOMO 0.35	102.7 KB	78.5 KB	18 ms	14 ms
YOLOv5	3.5 MB	1.9 MB	128 ms	81ms

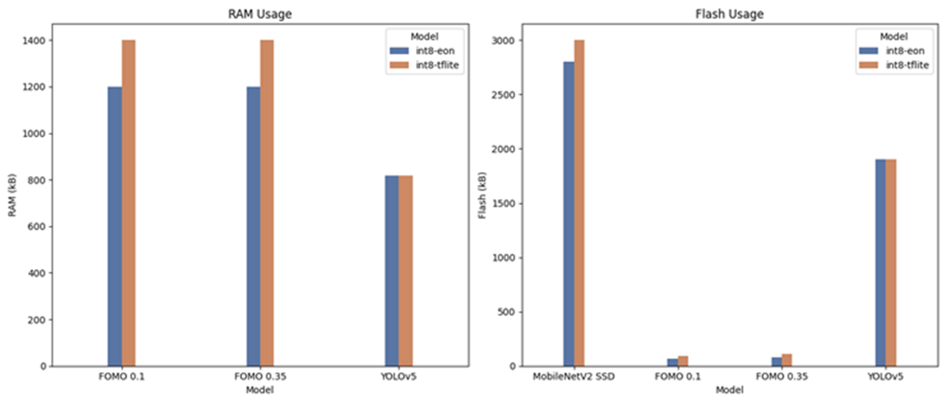


Fig. 15. Comparison of EON and TensorFlowLite models

Table 6. Review of memory utilization for EON and TensorFlow Lite models

	RAM (EON)	RAM (TensorFlowLite)	Flash (EON)	RAM (TensorFlowLite)
MobileNetV2 SSD	/	/	2.8 MB	3 MB
FOMO 0.1	1.2 MB	1.4 MB	64.4 KB	94.4 KB
FOMO 0.35	1.2 MB	1.4 MB	110.5 KB	78.5 KB
YOLOv5	817.5 KB	817.5 KB	1.9 MB	1.9 MB

potential loss of model accuracy resulting from optimization, an evaluation of the TensorFlow Lite model is provided using mAP (mean Average Precision) metric for evaluation. The analysis of the results concludes that the differences in accuracy between TensorFlow

Table 7. Evaluation of the TensorFlowLite model

	TensorFlow model	TensorFlow Lite model
AP	0.4535	0.4426
AP50	0.81	0.8
AP75	0.48	0.46
APs	0.003	0.0027
APm	0.35	0.3
APl	0.66	0.65
ARmax1	0.08	0.08
ARmax10	0.5	0.5
ARmax100	0.58	0.54
ARs	0.06	0.045
ARm	0.55	0.49
ARl	0.75	0.71
AP_car/	0.45	0.44

and TensorFlow Lite models are nearly indistinguishable. The TensorFlow model's precision is 45.35 %. At a 50 % overlap threshold, the model's precision is 81.3 %, while for a 75 % overlap threshold, the value is nearly halved. Detection precision for different sizes (APs – small, APm – medium, APl – large) indicates that the model performs well with larger objects. Additionally, the model exhibits better accuracy when there are multiple objects in the image (ARmax10, ARmax100). There is room for improvement in detecting small objects and under very strict overlap criteria.

EfficientDet 0 and 2 versions were deployed on the RPi device, and a comparison of inference time and model size is given in Figure 16. Memory analysis during the object detection process was performed and the results are given in Table 8. Both models require a similar amount of memory, with only minor differences in usage. EfficientDet 0 consumes slightly fewer resources than EfficientDet 2; however, both models function effectively within the available system resources. Through comprehensive testing

Table 8. Memory utilization of EfficientDet models (in KB)

	total	used	free	shared	buff/cache	available
EfficientDet 0	3794	599	1937	75	1256	3043
EfficientDet 2	3794	613	1915	80	1265	3025

and evaluation of NN models created using various tools, significant advancements in technologies for deploying models on edge devices have become apparent. By leveraging advanced optimization techniques, it is feasible to execute complex model architectures on resource-constrained devices. For example, models trained on Edge Impulse occupy

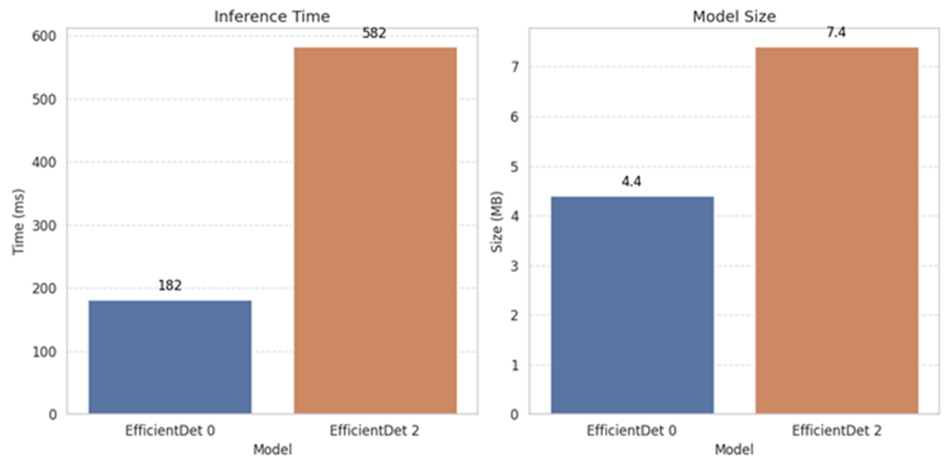


Fig. 16. Comparison of EfficientDet 0 and 2

considerably less space and consume fewer resources compared to TensorFlow Lite models. Conversely, the strength of the TensorFlow Lite framework lies in its broader selection of model architectures and the ability to utilize established models for object detection.

5. Conclusions

The proposed approach and experimental evaluations provide valuable insights into the challenges and opportunities of deploying DNNs for traffic object detection and recognition across the edge-fog computing continuum. One of the key takeaways is that distributing tasks between edge devices and fog servers offers a balanced trade-off between performance, resource consumption, and accuracy. By leveraging edge devices like smartphones for object detection, we can reduce data transmission and latency. We found that current smartphones perform well in both accuracy and speed, with only 33% CPU utilization during inference. This raises the question of whether offloading detection tasks to the fog is necessary, given that the communication overhead may outweigh any potential performance benefits. The results suggest that, in many cases, performing both detection and recognition on the mobile device itself is a more efficient strategy, especially for real-time applications.

Meanwhile, the fog server, with its higher computational capacity, is well-suited well-suited for more complex tasks such as car model recognition, where detailed feature extraction and processing are required. This delegation not only reduces the computational burden on edge devices like smartphones or embedded platforms but also enables the deployment of heavier models that might otherwise exceed the capabilities of smaller devices. For instance, running computationally intensive architectures such as YOLOv5 or EfficientDet on the fog server ensures that these models can operate without compromising performance or requiring extensive optimization, as would be necessary on edge devices. This division of labor improves the system's overall efficiency by allowing lightweight tasks, such as object detection, to occur locally on edge devices while offloading more demanding tasks to the fog server. Consequently, the combination of fast local inference with sophisticated fog-based recognition creates a robust pipeline that balances speed and accuracy across different layers of the edge-fog continuum.

While the smartphone proved capable, the resource constrained embedded devices, such as RPi or Arduino Nano, could still benefit from fog or cloud offloading, especially for complex models. However, our focus in this study was on evaluating lightweight architectures directly on these devices. Future research could explore hybrid approaches, where RPi or Arduino devices collaborate with fog or cloud systems for more demanding tasks, striking a balance between local processing and offloading. We found that the quantized and minimized model deployed on Arduino Nano, RPI device, and Android smartphones achieved reasonable performance with efficient resource usage. The use of quantization and model optimization techniques, especially through TensorFlow Lite and Edge Impulse, proved crucial for deploying DNNs on resource-constrained devices like the Arduino Nano and RPi. Our results show that while Edge Impulse models consume fewer resources, TensorFlow Lite offers greater flexibility through access to a wider variety of model architectures. The experiments also highlighted the trade-offs between model size, inference time, and accuracy, emphasizing the importance of fine-tuning hyperparameters based on the specific hardware and use case.

The ability to achieve near real-time detection using optimized models demonstrates that modern AI technologies can effectively run on low-power edge devices. Our findings suggest that combining edge and fog computing provides a scalable and efficient way to implement more demanding AI solutions, such as specific object recognition on video stream. The experiments also revealed that model size and computational overhead must be carefully managed, particularly on microcontrollers, where even slight increases in image resolution or model complexity can lead to significant resource constraints.

However, there are still several avenues for future research in this area. Some potential directions include:

- Exploration of federated learning techniques tailored for object detection and recognition tasks at the edge for aggregating model updates from distributed edge nodes efficiently while accounting for resource constraints.
- Investigation of online continual learning techniques for adaptive object detection and recognition at the edge that enable edge devices to incrementally learn from streaming data while retaining knowledge learned from previous tasks.
- Research resource-efficient model adaptation techniques in edge-fog environments that dynamically adjust model complexity and capacity based on available computational resources and streaming data characteristics.

By further exploring these research directions, we can continue to advance the field of DNN model distribution and slicing across the edge-fog-cloud computing continuum, enabling resource-aware and efficient object detection, classification and recognition systems for various real-world applications.

Acknowledgments. Research presented in this paper is supported by the Ministry of Science and Technological Development, Republic of Serbia, 451-03-65/2024-03/200102, Erasmus+ Project FAAI: The Future is in Applied AI (WP4 - Artificial Intelligence framework for training in HE)

- 2022-1-PL01-KA220-HED-000088359 and COST Action CERCIRAS - Connecting Education and Research Communities for an Innovative Resource Aware Society - CA19135.

References

1. Akhtar, A., Ahmed, R., Yousaf, M.H., Velastin, S.A.: Real-time motorbike detection: Ai on the edge perspective. Mathematics 12(7) (2024), `https://www.mdpi.com/2227-7390/12/7/1103`
2. Berwo, M.A., Khan, A., Fang, Y., Fahim, H., Javaid, S., Mahmood, J., Abideen, Z.U., M.S., S.: Deep learning techniques for vehicle detection and classification from images/videos: A survey. Sensors 23(10) (2023), `https://www.mdpi.com/1424-8220/23/10/4832`
3. Bian, J., Arafat, A.A., Xiong, H., Li, J., Li, L., Chen, H., Wang, J., Dou, D., Guo, Z.: Machine learning in real-time internet of things (iot) systems: A survey. IEEE Internet of Things Journal 9(11), 8364–8386 (2022)
4. Bittencourt, L., Immich, R., Sakellariou, R., Fonseca, N., Madeira, E., Curado, M., Villas, L., DaSilva, L., Lee, C., Rana, O.: The internet of things, fog and cloud continuum: Integration and challenges. Internet of Things 3-4, 134–155 (2018)
5. Cho, E., Yoon, J., Baek, D., Lee, D., Bae, D.: Dnn model deployment on distributed edges. In: Bakaev, M., Ko, I.Y., Mrissa, M., Pautasso, C., Srivastava, A. (eds.) ICWE 2021 Workshops. Communications in Computer and Information Science, vol. 1508. Springer, Cham (2021)
6. Dharani, A., Kumar, S.A., Patil, P.N.: Object detection at edge using tinyml models. SN Computer Science 5(1), 11 (11 2023), `https://doi.org/10.1007/s42979-023-02304-z`
7. Hanhirova, J., Kämäräinen, T., Seppälä, S., Siekkinen, M., Hirvisalo, V., Ylä-Jääski, A.: Latency and throughput characterization of convolutional neural networks for mobile computer vision. In: Proceedings of the 9th ACM Multimedia Systems Conference (MMSys '18). pp. 204–215. New York, NY, USA (2018)
8. Hymel, S., Banbury, C.R., Situnayake, D., Elium, A., Ward, C., Kelcey, M., Baaijens, M., Majchrzycki, M., Plunkett, J., Tischler, D., Grande, A., Moreau, L., Maslov, D., Beavis, A., Jongboom, J., Reddi, V.J.: Edge impulse: An mlops platform for tiny machine learning. ArXiv abs/2212.03332 (2022), `https://api.semanticscholar.org/CorpusID:254366602`
9. Iftikhar, S., Gill, S.S., Song, C., Xu, M., Aslanpour, M.S., Toosi, A.N., Du, J., Wu, H., Ghosh, S., Chowdhury, D., Golec, M., Kumar, M., Abdelmoniem, A.M., Cuadrado, F., Varghese, B., Rana, O., Dustdar, S., Uhlig, S.: Ai-based fog and edge computing: A systematic review, taxonomy and future directions. Internet of Things 21, 100674 (2023), `https://www.sciencedirect.com/science/article/pii/S254266052200155X`
10. Kallimani, R., Pai, K., Raghuwanshi, P., Iyer, S., López, O.L.A.: Tinyml: Tools, applications, challenges, and future research directions. Multimedia Tools and Applications 83(10), 29015–29045 (2024), `https://doi.org/10.1007/s11042-023-16740-9`, dOI: 10.1007/s11042-023-16740-9
11. Lee, J.K., Varghese, B., Woods, R., Vandierendonck, H.: Tod: Transprecise object detection to maximize real-time accuracy on the edge. In: Proceeding of the 5th EEE 5th International Conference on Fog and Edge Computing. pp. 53–60 (2021)
12. Lin, C.Y., Wang, T.C., Chen, K.C., Lee, B.Y., Kuo, J.J.: Distributed deep neural network deployment for smart devices from the edge to the cloud. In: Proceedings of the ACM MobiHoc Workshop on Pervasive Systems in the IoT Era. p. 43–48. PERSIST-IoT '19, Association for Computing Machinery, New York, NY, USA (2019), `https://doi.org/10.1145/3331052.3332477`

13. Lockhart, L., Harvey, P., Imai, P., Willis, P., Varghese, B.: Scission: Performance-driven and context-aware cloud-edge distribution of deep neural networks. In: Proceedings of the IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC). pp. 257–268. Leicester, United Kingdom (2020)
14. Marszałek, M., Schmid, C.: Inria annotations for graz-02 dataset (2007), `https://lear.inrialpes.fr/people/marszalek/data/ig02/`, accessed: 2024-10-1
15. McNamee, F., Dustdar, S., Kilpatrick, P., Shi, W., Spence, I., Varghese, B.: The case for adaptive deep neural networks in edge computing. In: Proceedings of the IEEE 14th International Conference on Cloud Computing (CLOUD). pp. 43–52 (2021)
16. Parthasarathy, A., Krishnamachari, B.: Defer: Distributed edge inference for deep neural networks. In: Proceedings of the 14th International Conference on COMmunication Systems & NETworkS (COMSNETS). pp. 749–753 (2022)
17. Ren, W., Qu, Y., Dong, C., Jing, Y., Sun, H., Wu, Q., Guo, S.: A survey on collaborative dnn inference for edge intelligence. Machine Intelligence Research 20, 370–395 (2023)
18. Shuvo, M.M.H., Islam, S.K., Cheng, J., Morshed, B.I.: Efficient acceleration of deep learning inference on resource-constrained edge devices: A review. Proceedings of the IEEE 111(1), 42–91 (2023)
19. Singh, R., Gill, S.S.: Edge ai: A survey. Internet of Things and Cyber-Physical Systems 3, 71–92 (2023), `https://www.sciencedirect.com/science/article/pii/S2667345223000196`
20. Stojanovic, D., Sentic, S., Stojanovic, N.: Towards resource-efficient dnn deployment for traffic object recognition: From edge to fog. In: Zeinalipour, D., Blanco Heras, D., Pallis, G., Herodotou, H., Trihinas, D., Balouek, D., Diehl, P., Cojean, T., Fürlinger, K., Kirkeby, M.H., Nardelli, M., Di Sanzo, P. (eds.) Euro-Par 2023: Parallel Processing Workshops. pp. 30–39. Springer Nature Switzerland, Cham (2024)
21. Taheri, J., Dustdar, S., Zomaya, A., Deng, S.: Edge Intelligence: From Theory to Practice. Springer International Publishing (2023), `https://books.google.es/books?id=cCV5zwEACAAJ`
22. Teerapittayanon, S., McDanel, B., Kung, H.T.: Distributed deep neural networks over the cloud, the edge and end devices. In: Proceedings of the IEEE 37th International Conference on Distributed Computing Systems (ICDCS). pp. 328–339. Atlanta, GA, USA (2017)
23. Udacity: Udacity self-driving car dataset (2024), `https://public.roboflow.com/object-detection/self-driving-car`, accessed: 2024-10-1
24. Yong-Hah, R.: Bike-rider detector (2019), `https://github.com/yonghah/bikerider-detector`, accessed: 2024-10-1

Dragan Stojanović is a professor in the Department of Computer Science, Faculty of Electronic Engineering, University of Nis. His research and development interests include mobile computing and Internet of Things, Big Data processing and analytics, edge intelligence and resource-aware computing over edge-cloud computing continuum.

Stefan Sentić completed Master degree at the Faculty of Electronic Engineering, University of Nis and now works as a software engineer at the HTEC company. His research and development interests include mobile applications and systems development, edge intelligence in Internet of Things.

Natalija Stojanović is a professor in the Department of Computer Science, Faculty of Electronic Engineering, University of Nis. Her research and development interests include Big Data processing and analytics, and high performance distributed, parallel and ubiquitous computing.

Teodora Stamenković completed Master degree at the Faculty of Electronic Engineering, University of Nis and now works as a software engineer at the Nignite company. Her research and development interests include mobile and ubiquitous computing, edge intelligence and the Internet of Things.

Received: May 03, 2024; Accepted: December 13, 2024.

Computer Science and Information Systems 22(2):649–671 https://doi.org/10.2298/CSIS240914028K

Energy-efficiency of software and hardware algorithms*

Maja H. Kirkeby[1], Thomas Krabben[1], Maria B. Mikkelsen[1], Mads Rosendahl[1], Mathias Larsen[2], Martin Sundman[2], Tjark Petersen[3], and Martin Schoeberl[3]

[1] Department of People and Technology, Roskilde University,
Roskilde, Denmark
{majaht,krabben, mariabm, madsr} @ruc.dk
[2] IT University of Copenhagen, Copenhagen, Denmark
{mathl,sund}@itu.dk
[3] DTU Compute, Technical University of Denmark,
Lyngby, Denmark
s186083@student.dtu.dk
masca@dtu.dk

Abstract. In this article, we compare the energy efficiency of hardware and software implementations of Heapsort and Dijkstra's algorithm for route finding. The software implementations are written in C for Raspberry Pi, and the hardware implementations are crafted in Chisel for an FPGA. Our objective is to examine how we can fairly compare energy efficiency between hardware and software. These solutions are positioned to replace each other in operational contexts, necessitating a comparison of their whole-system energy consumption. This study seeks to identify circumstances where time and energy efficiency diverge, offering insights to guide hardware selection. Our findings serve as a step towards understanding the complex trade-offs in algorithm performance across different computational platforms.

Keywords: energy efficiency, performance, FPGA, CPU, algorithms

1. Introduction

Improving software's time and energy efficiency is essential [1, 6]. Implementing the algorithms in Field-Programmable Gate Arrays (FPGAs) demonstrates substantial performance gains in highly parallelizable tasks, e.g., [18, 30]. However, only a few have considered the energy efficiency of hardware implementations [14, 17]. Previous studies on improving algorithms using FPGAs have employed FPGAs in different ways. One approach which requires specialized expertise is to implement the algorithm in hardware-specifying languages such as Verilog or VHDL [28, 30]. Alternative approaches, oriented toward software developers, use FPGAs as accelerators for, e.g., C/C++ programs [14] or, as in this study, where use the high-level programming language Chisel [3, 27].

* This work is supported by the Innovation Fund Denmark for the project DIREC (9142-00001B), by CERCIRAS Cost Action CA19135, by the Independent Research Fund Denmark Project no. 2102-00281B, and is based on work from the COST Action CERCIRAS CA19135, supported by COST (European Cooperation in Science and Technology).

We will compare two alternative solutions, i.e., a hardware and a software solution, in operational contexts. We will evaluate the energy efficiency and performance of two computing platforms (running the algorithmic implementations) that may replace each other in operational contexts. Our work extends an initial studies presented and published at the 2022 Workshop on Resource Awareness of Systems and Society [18], which is reported in Section 7. This work indicated an exciting upper bound for performance improvement when comparing a highly parallelizable program, Conway's Game of Life [4], to software implementations. Although an upper bound is interesting, it does not provide any insight into the gain from converting ordinary software implementations into hardware implementations for ordinary programs. In this paper, we will focus on providing a more well-rounded comparison, studying algorithms where neither software nor hardware has particular advantages:

How do time and energy consumption compare for hardware and software implementations of ordinary software algorithms?

In our choice of algorithms, it is our aim that it should not provide any particular advantage for either hardware or software versions. It is possible to find highly parallelizable algorithms where the hardware implementation will have an obvious advantage. In our experiments, we use Heapsort and Dijkstra's algorithm to find the shortest path in a graph. Both are widely used and have well-known and studied implementations. Both algorithms are not easy to parallelize [10, 13], but can still contain a fair amount of parallelism at the local level.

The main contributions of this article are:

1. *Comparative Analysis:* The paper compares the energy efficiency between software and hardware implementations of two well-known algorithms: Heapsort and Dijkstra's algorithm. This comparison is new in that it focuses on both energy consumption and performance across software and hardware implementations and, thus, provides the first insights into the energy and performance trade-offs
2. *Methodological Innovation:* It introduces a methodological framework that ensures fair comparisons between software and hardware implementations. This includes a thorough discussion of the choice of measurement techniques.

In the following, we provide the basis for a fair comparison (Section 2), and in Section 3, we introduce the specifications and implementations of the algorithms. Section 4 describes our experimental setup, and Section 5 describes our results. Section 6 discusses the results. Section 7 compares implementations of the Game-of-Live to explore an upper bound of possible speedups and energy savings in an FPGA. Afterward, we discuss the related work (Section 8) and contextualize our results. Section 9 concludes the paper and summarizes future work.

2. A Fair Comparison

In this section, we discuss a methodological framework to ensure that comparisons between software and hardware implementations are fair and insightful. By detailing the choice of hardware and measurement protocols, we highlight the practical considerations underlying the comparison, providing a foundation for the detailed descriptions of the specific implementations that follow.

2.1. Choice of Measurement

Previous work on energy consumption of software implementations has employed energy estimations using Intel's Running Average Power Limit (RAPL) [11], as it has been reported as having negligible overhead and providing precise results [9, 24]. e.g., [23]. However, while RAPL is precise and highly correlated (a value of 0.99) [16]) with the actual power dissipation, it does not provide accurate results, i.e., the RAPL measurements are not close to the true energy consumption. Thus, instead, we could employ external measurements that provide each systems's ground truth energy consumption. This choice also has drawbacks. For instance, it introduces more noise since it measures the energy consumption of the entire device compared to only the CPU. It also introduces an imprecision in the synchronization between the time in the measuring unit obtaining the energy consumption and the time in the measured device that executes the algorithm.

In this study, the two computing platforms running the algorithmic implementations are positioned to replace each other in operational contexts. Therefore, we employ external energy measurements to capture the energy consumption of the entire systems. This approach allows for a fair comparison across platforms by reflecting the energy impact of platform-specific overheads, such as memory and I/O subsystems, which are integral to real-world operation. While external measurements introduce noise and potential synchronization imprecision, they provide a more accurate and comprehensive evaluation of whole-system energy consumption.

2.2. Choice of Hardware

One factor that influences the energy consumption of both hardware and software implementations is the choice of hardware. Since the hardware is very different in the two cases, we will use the same requirement for choosing the hardware, namely, to use cheap and available hardware.

There are many cheap and available computers for software implementations, e.g., Orange Pi, Asus Tinker, Nvidia Jetson Nano, or Raspberry Pi. However, in this study, we have chosen a standard Raspberry Pi 4 computer Model B with 4GB RAM and a 1.5 GHz 64-bit quad-core ARM Cortex-A72 processor running the standard Raspberry Pi OS, a Linux version.

There are two types of FPGA units: pure FPGA boards and system-on-chip FPGA boards. Choosing the pure FPGA board will allow us to measure the exact energy consumption of the FPGA unit alone without interference from other processors, which would be the case with System on Chip FPGA boards.

For the hardware implementations, we have chosen a Digilent FPGA board Cmod A7 (version Cmod A7-35T) with an XC7A35T-1CPG236C FPGA unit, an MSPS Onchip ADC, 20800 Look-up Tables (LUTs), 41600 Flip-Flops, 225 KB Block RAM and 5 Clock Management Tiles.

2.3. Choice of Implementations

In addition, previous studies, e.g., [6, 8, 23], highlight that whole-systems energy consumption provides insights into the varying energy consumption across different implementations, illustrating how choices in software development impact overall energy use. Thus, for instance the choice of programming language [23], implementation style [8], and compiler flags [26] influence the energy consumption and execution time of the programs. For the hardware implementation, energy factors typically include the number of logic elements and routing resources, see, e.g., [29].

For the software solutions, we base it on standard C-implementations, e.g., following the descriptions in [5], using standard settings of optimizations in the GNU GCC compiler. The Hardware solutions are hand-written versions in Chisel with common approaches to optimize the design for improved performance. How we have realized the algorithms in hardware is discussed in the next section.

To ensure fairness, we align the implementations of the algorithms across both platforms. This alignment ensures that the differences in energy consumption and execution time reflect the platforms' inherent characteristics rather than variations in algorithmic design. The implementations will:

1. carry out Dijkstra's algorithm and Heapsort, respectively.
2. avoid external communication by including the algorithm inputs in the implementation instead of reading the input from external files.
3. return the smallest subset of the result to ensure computation of the results while reducing the read/write accesses.

This standardized approach supports meaningful comparisons while adhering to the operational requirements of each platform.

3. Algorithms

This section provides detailed descriptions of Heapsort and Dijkstra's implementations. With a focus on implementation specifics, the next Section 4 allows us to finalize the experimental design before providing the results in Section 5.

3.1. Heapsort: Software

The implementation of Heapsort uses a standard implementation from the Rosetta repository[4], see Figure 1. It uses a max-heap data structure, storing values in a balanced tree where a node is bigger than its children. The tree is represented as an array. In a binary tree, the children at array index i can be found at array index $i * 2 + 1$ and $i * 2 + 2$. The implementation uses a k-heap with slightly better complexity measures since the tree will have a smaller depth for the same number of store values.

The k-heap structure is initially established by moving values down the tree structure if they are smaller than their children. In the second phase, the biggest value is repeatedly moved to the back of the array, and the heap structure is reestablished.

[4] `http://www.rosettacode.org/`

```
// heapsort start
int max (int *a, int n, int parent) {
  int largest = parent;
  for (int child = (K*parent)+1; child < (K*parent)+K+1; child++)
    if (child < n && a[child] > a[largest]) largest = child;
  return largest;
}
void downheap (int *a, int n, int i) {
  while (1) {   int j = max(a, n, i);
                if (j == i) break;
                int t = a[i];
                a[i] = a[j];
                a[j] = t;
                i = j;
  }
}
void heapsort (int *a, int n) {
  int i;
  for (i = (n - 2) / K; i >= 0; i--) downheap(a, n, i);
  for (i = 0; i < n; i++) {
    int t = a[n - i - 1];
    a[n - i - 1] = a[0];
    a[0] = t;
    downheap(a, n - i - 1, 0);
  }
} // heapsort end
```

Fig. 1. The software implementation of Heapsort in C, where `a` is the input array to be sorted and `n` is its size

3.2. Heapsort: Hardware

The hardware solution of Heapsort uses higher order k-max-heaps to increase parallelism. In the k-heap, $k+1$ elements must be compared in each step while re-establishing the heap order. In hardware, this comparison can be done in parallel, thus theoretically allowing for the heap order to be established in $\log_k(n)$ clock cycles. Practically, fetching all required values from memory, finding the largest of them, and swapping the parent and the largest child if the heap order is violated has to be spread over multiple clock cycles to allow the circuit to be operated at high clock frequencies. An architectural diagram of the heap

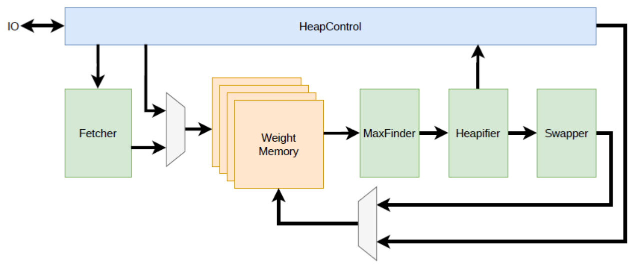


Fig. 2. The architecture of the hardware implementation for Heapsort

module is shown in Figure 2. The circuit can not easily be pipelined since deciding which child should be the next parent while traversing the heap downwards is always delayed, resulting in a pipelined implementation having to stall most of the time.

Nonetheless, parallelism can be exploited in multiple instances, giving the hardware implementation an edge over a software implementation in clock cycles per iteration. All k children of a given node can be fetched simultaneously using wide memories, which store k concatenated values per address. Thus, only two load operations are needed per iteration. The maximum between the parent and all k children nodes can be found using a tree of blocks, where the maximum of two values is selected. The total delay of this circuit is $\log_2(k)$ times the delay of a single block. For larger values of k, the tree has to be divided into multiple levels separated by registers to allow for operation at high clock frequencies. The write operations to the memory associated with a swap can be overlapped with fetching the next parent or children, depending on the direction of the traversal.

The described heap module for sorting is paired with a small circuit that inserts values from memory into the heap and then extracts the ordered sequence from the heap by continuously removing the root until the heap is empty.

3.3. Dijkstra: Software

The implementation of Dijkstra's algorithm represents the graph as an adjacency list, where all edges from a vertex are grouped together. The algorithm computes the shortest route from a given vertex (here, the vertex at index 0) to all other vertices. For each vertex, it will find the previous vertex in the shortest route from the start to that vertex and the weight of that route. The actual route can be found by following the previous vertex indices through the graph until the start vertex is reached. The implementation can be seen in Figure 3.

3.4. Dijkstra: Hardware

A priority queue-based system allows for an easy determination of the next node to visit. Using the hardware heap of the heap sort experiment, a hardware priority queue could be constructed that accepts a new value worst case every $\log_k(n)$ clock cycles, where n is the number of nodes in the graph and k is the degree of parallelism in the heap. This results in $n(n-1)d\log_k(n)$ clock cycles to execute Dijkstra's algorithm, where d is the density of the graph with $d = 1$ representing a fully connected graph. This solution is not easily parallelizable since it would require a priority queue that can insert multiple values simultaneously.

An alternative implementation uses a linear search through all nodes to determine the next node to visit. This proves to be an advantage since the route updates and search for the next node to visit can be conducted in parallel in hardware. The length of an alternative path through the currently visited node is calculated for each node. If the alternative path is shorter, it is written to the route table instead of the previously known distance from the start. At the same time, a state element holding the closest unvisited node yet to be encountered is updated if this node is closer to the start. After all nodes have been visited, the state element holds the next node to visit. This results in $n(n-1)$ clock cycles to execute Dijkstra's algorithm. This solution can easily be parallelized by working

```
// the graph is stored as an adjacency list where edges are
// ordered by start vertex input to Dijkstra's algorithm
int n;
#define n 6 // number of vertices
#define m 9 // number of edges
int node1[]={ 0, 0, 0, 1, 1, 2, 2, 3, 4}; // edges' start vertex,
int node2[]={1, 2, 5, 2, 3, 3, 5, 4, 5};  // edges' end vertex
int dist[]={7, 9, 14, 10, 15, 11, 2, 6, 9};     // edges' weight
int edge[]={0, 3, 5, 7, 8, 0};        // vertex to edge index link
// Data structures for the algorithm
int done[n], prev[n], wght[n];
const int MAX=1000000;
void main(){
  int start = 0;
  // Dijkstra start
  for(int i=0;i<n;i++){ done[i]=0; prev[i]=-1; wght[i]=MAX;}
  wght[start]=0;
  int cur=-1, w = 0;
  for(int k=0;k<n;k++) {
    cur = -1; w = MAX;
    for (int i = 0; i < n; i++)
      if (done[i]==0 && wght[i] < w) { cur = i; w = wght[i];}
    if (cur<0) break;
    int j = edge[cur];
    while (j < m) {
      int n1 = node1[j], n2 = node2[j], d = dist[j];
      if (n1 != cur) break;
      int d2 = wght[n2], d3 = w + d;
      if (d2 > d3) { prev[n2] = cur; wght[n2] = d3;}
      j++;
    }
    done[cur] = 1;
  }
  cur = n-1;
  // Dijkstra end
}
```

Fig. 3. Dijkstra's shortest path algorithm; it finds the path from vertex 0

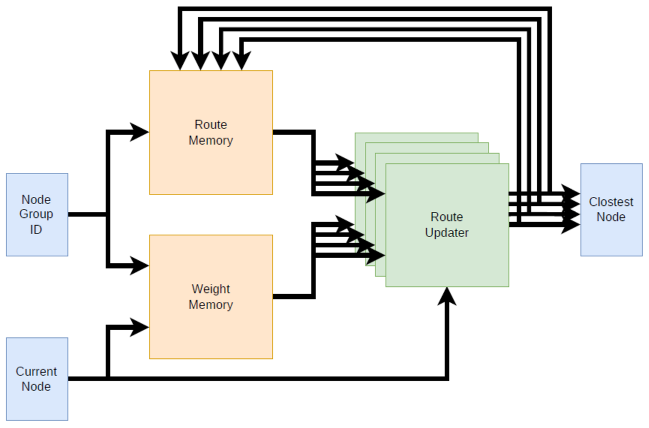


Fig. 4. The architecture of the hardware implementation of Dijkstra's algorithm

on multiple path updates simultaneously. This requires multiple read ports on the weight memory and multiple read and write ports to the route table. Furthermore, not only one route has to be compared to the closest unvisited not yet to be encountered but multiple at the same time.

The second solution is chosen over the first since it is easily parallelizable while requiring significantly fewer hardware resources. This translates into lower power dissipation and only worse performance on graphs of density below $1/log_k(n)$. Considering the total energy of the execution of Dijkstra's algorithm, the density at which both solutions perform equally well is offset even more in favor of the second solution since its lower power consumption compensates for the longer run time.

An architectural diagram of the hardware implementation of Dijkstra's algorithm is shown in Figure 4. A group of k nodes reads their preliminary shortest path, their visited status from the route memory, and their weight to the currently visited node from the weight memory. For each node in the group, the distance of a new route through the currently visited node is calculated and compared to the old shortest path. The shorter of the two routes is selected and written to the route memory. Furthermore, the state element holding the closest unvisited node to the start is updated if the distances sent to the route memory are shorter. In the hardware implementation, the design is pipelined with the two memory modules separating the circuit into two stages.

To support single-cycle access to the weights without using $O(n2)$ memory, a buffering solution that exploits the known direction of traversal of the weights is employed. This solution stores weights as packed, unaligned adjacency lists, thus only requiring $O(e)$ memory, where e is the number of directed edges in the graph.

4. Experimental Setup

The hardware implementations were executed on a Digilent FPGA board Cmod A7 (version Cmod A7-35T), and the software implementations were compiled with gcc (Raspbian 8.3.0-6+rpi1) 8.3.0 with flag `-O2` and executed on a Raspberry Pi 4 computer Model B 4GB RAM. There is one program per input array, i.e., one file per software and hardware implementation and input array.

4.1. Input spaces

In Heapsort, the runtime and, therefore, perhaps also the energy consumption depend on the number of input elements and their order. A previous study [17] showed that while the order matters for runtime, the comparability arises from using the same order in all experiments. In this study, we (1) always use the same order, namely ordered input, which causes the highest number of comparisons, and (2) vary the number of elements: 4096, 6144, 8192, 10240, 12288, 14336, and 16384.

For Dijkstra's shortest path algorithm, the execution time depends on the type of graph, i.e., sparse or dense and directed or undirected, and the start node for which the path is calculated. In the sorting algorithm, the focus was on growth, and for the shortest path algorithm, we varied only the form and the starting node. The implementation will calculate routes switching between all the possible start vertices.

4.2. Pre-study: Configuration of Heapsort

An exploratory study [17] found that the optimal degree of parallelism in hardware programs differs when considering energy and time. The kind of parallelism in hardware algorithms and software algorithms differs, as described in Section 3, and, thus, the optimal degree of parallelism differs for software and hardware implementations and differs for each algorithm. A fair comparison allows the optimal degree of parallelism, and therefore, we have run a pre-study to find the optimal degree of parallelism. For both hardware and software implementations, we have optimized for energy. Both Dijkstra and Heapsort can have different degrees of parallelism in the hardware implementation, while for the software implementation, this is only true for Heapsort.

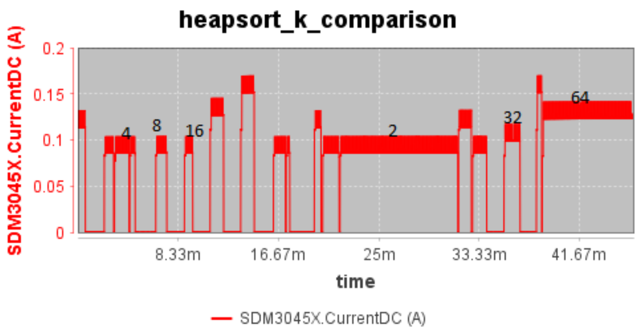


Fig. 5. The current over time for Heapsort using different $k \in \{2, 4, 8, 16, 32, 64\}$ on FPGA inputs of size 4096. The k-values are not tested in order; the k-values are annotated above the associated execution. In the graph, we also see error-prone executions without annotations. The $k = 16$ provides the energy optimal execution

Hardware The test-run graph obtained directly from the Siglent can be seen in Figure 5[5]. It shows the current over time (in minutes) for Heapsort using different k-values, i.e., $k \in \{2, 4, 8, 16, 32, 64\}$ on FPGA inputs of size 4096.

The k values are not tested in order and are annotated above the associated execution. In the graph, we also see error-prone executions without annotations. The $k = 16$ provides the energy-optimal execution since it has the least area under the curve, i.e., the shortest execution time and the lowest current (the voltage is fixed).

Software The growth rate is almost linear, as expected from an algorithm that runs at $n \log_k n$, see Figure 6. Interestingly, the constant factor does matter since the fastest time is with $k = 4$. For bigger k, finding the subtree with the biggest root note becomes the main bottleneck in the execution.

[5] The original experimental measure data was lost, and we cannot provide precise energy consumption.

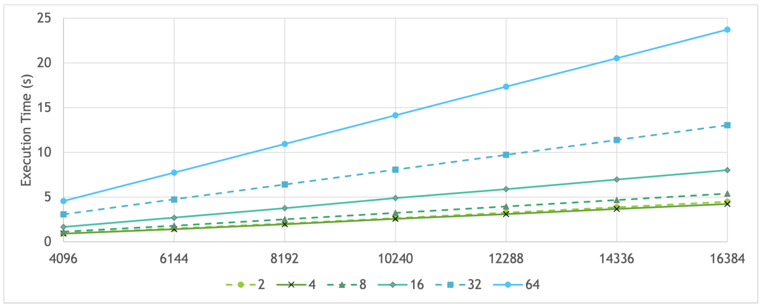


Fig. 6. The average execution time in seconds for Heapsort using a k-heap on Raspberry Pi over inputs from 4096 to 16384. The $k = 4$ provides the fastest executions

4.3. Measuring and Adjustment of Repetitions

We measure the energy consumption for the entire Raspberry Pi using a programmable power supply Siglent SPD3303X-E Linear DC 3CH in connection with a Siglent SDM30-45X Digital Multimeter is a 4 1/2 digit (66,000 count) multimeter. This setup allows for the required high-precision readings of the current. The equipment spans the current of both the FPGA and the Raspberry Pi and allows us to measure the power dissipation each 100ms.

```
#define iter 500000
int main(){
  int max1=iter/n, lng=0;
  for(int n1=0;n1<n;n1++){
    for(int k1=0;k1<max1;k1++){
    int start=n1;
    // Dijkstra start ... end
    while(cur!=start){
      lng+=wght[cur];
      cur = prev[cur];
    }
    printf("done %d\n",lng);
    }
  }
  return 0;
}
```

(a) Dijkstra

```
#define iter = 3000
// Heapsort start ...  end
int main () {
int i, j:
  for (i = 0; i < iter; i++) {
    for (j = 0; j < N; j++){
        a[j] = j;}
    heapsort(a, N);
    printf("done %d\n",a[0]);
  }
  return 0;
}
```

(b) Heapsort

Fig. 7. Dijkstra's shortest path software algorithm (Figure 7a) will calculate 500.000 routes switching between the n possible start vertices and the Heapsort software algorithm will iterate its sorting 3000 times

In this experiment, we measure power continuously and the total execution time a single run for each input size. We want to calculate an average power dissipation and average execution time for one execution of the algorithm for each input size. Because the algorithms executes very fast, we cannot measure a single execution. To ensure a large enough sample size of the current, we adjust the total execution time to be at least 3 seconds by adjusting the number of iterations. The adjustments occur within both the software and hardware implementation, see Table 1 for a precise number of iterations used in the hardware and software implementations. The adjustment setup within the software implementations differ, see Figure 7b for adjustments used for Heapsort and Figure 7a for the adjustments used for Dijkstra. This adjustment will also reduce the synchronization imprecision between the equipment and the hardware.

Table 1. Overview of the number of in-algorithm iterations that ensure a least execution time of 3 seconds

Algorithm	**Mode**	**input size(s)**	**Iterations**
Heapsort	Software	all	3000
	Hardware	4096	1550
		6144	1033
		8192	775
		10240	620
		12288	516
		14336	442
		16384	387
Dijkstra	Software	all	500000
	Hardware	all	9000

From the measured power, we calculate an average power dissipation, and using the number of iterations, see Table 1, and the measured total execution time, we calculate the average execution time for a single iteration.

5. Results

Our results demonstrate significant and contrasting differences in energy efficiency and performance between Heapsort and Dijkstra's software and hardware implementations. These findings may necessitate more nuanced insights into the optimal hardware selection based on the algorithmic demands.

5.1. Heapsort

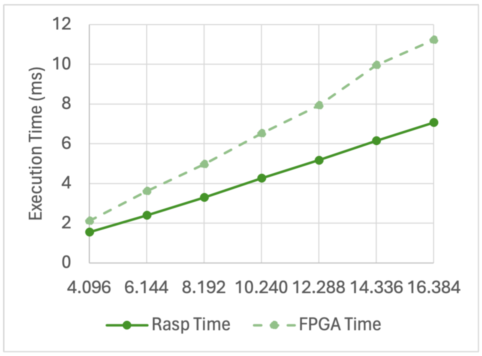
Comparative data on Heapsort's time and energy consumption in an FPGA and the Raspberry Pi is shown in Figure 8 and Table 2, with a focus on the energy efficiency achieved through hardware implementation. These results demonstrate energy savings and extra time costs when employing FPGAs over traditional CPUs. It is worth noticing that while the power dissipation by software and hardware implementations are different, they seem constant for the individual implementation.

Table 2. Heapsort's energy consumption within FPGA (k=16) and Raspberry Pi (k=4)

Input size	Raspberry Pi Time pr. iter. (ms)	Raspberry Pi Power (W)	Raspberry Pi Energy pr. iter. (mJ)	FPGA Time pr. iter. (ms)	FPGA Power (W)	FPGA Energy pr. iter. (mJ)
4096	1.541	3.043	4.689	2.102	0.431	0.906
6144	2.383	3.050	7.269	3.614	0.571	2.062
8192	3.296	3.050	10.055	4.959	0.572	2.837
10240	4.259	3.032	12.913	6.513	0.573	3.731
12288	5.166	3.047	15.741	7.922	0.572	4.533
14336	6.150	3.064	18.847	9.944	0.569	5.656
16384	7.066	3.080	21.764	11.213	0.378	4.241

The energy consumption of the Heapsort implementation on the Raspberry Pi is highly correlated to the execution time. The time consumption follows the expected $n(\log n)$ time complexity, and there is no significant increase in energy consumption when larger parts of memory are used during execution.

The total execution times are greater than 3 seconds, see Table 3; the total executions times are not directly comparable because the number of iterations within each run differ, see Table 1. Table 3 also shows a 2-second difference between the Siglent and the Raspberry time measurements. This is because we have chosen to count an experiment as when the Raspberry current increases beyond a certain threshold; in this case, when the current reaches 0.55A and more. This methodology cuts the execution time short on both ends. The energy consumption is calculated based on the logged times from the Raspberry Pi and the average power dissipation. While the execution time becomes correct, this method increases the average power dissipation slightly, and thus, we report a slightly larger energy consumption for the Raspberry Pi.



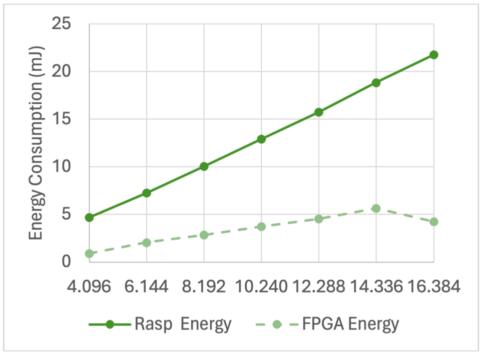


Fig. 8. Heapsort's time and energy consumption within FPGA (k=16) and Raspberry Pi (k=4)

Table 3. Heapsort's total execution time (s) per input. They are not directly comparable because the number of repetitions differ

Input size	**Raspberry Pi**		**FPGA**
	(Siglent)	(CPU)	
4096	2.536	4.623	3.258
6144	5.017	7.15	3.733
8192	7.712	9.888	3.843
10240	11.388	12.777	4.038
12288	13.672	15.499	4.088
14336	16.495	18.450	4.395
16384	19.095	21.198	4.34

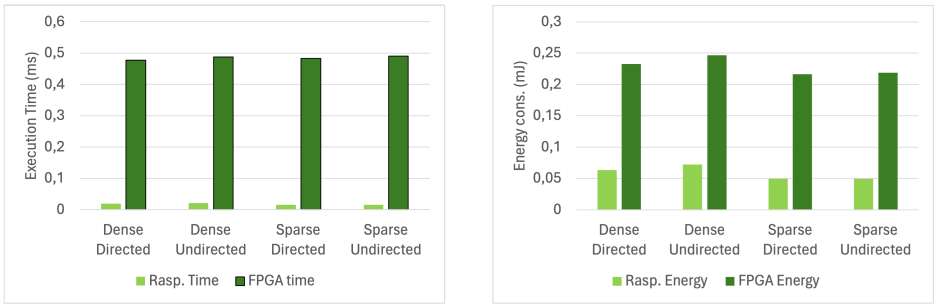


Fig. 9. Dijkstra: Avg. Time and Energy consumption within FPGA and Raspberry Pi

5.2. Dijkstra

The results show that Dijkstra's shortest path algorithm does not necessarily benefit from hardware implementations when considering performance or energy, see Figure 9. The FPGA platform demonstrates higher average times and energy consumption per iteration than the Raspberry Pi across various graph configurations. For example, in scenarios like dense directed graphs, it consumed 0.232 mJ per iteration compared to 0.064 mJ by the Raspberry Pi, demonstrating a significantly higher energy usage, see Table 4.

These results suggest that while FPGAs are typically considered for their potential to enhance performance through parallelism, their energy efficiency for Dijkstra's algorithm is less effective than traditional processing on a Raspberry Pi. This revelation is particularly crucial for applications where energy consumption is as critical as processing speed, such as in portable or embedded devices where power conservation is essential.

6. Discussion of Results

Our results reveal new trade-off considerations between algorithm efficiency and performance. This discussion contextualizes our findings, leading directly to Section 9 where we summarize the key insights and summarize avenues for further research.

Table 4. Dijkstra's average time and energy consumption within FPGA and Raspberry Pi

	Raspberry P			**FPGA**		
Experiment	Time pr. iter. (ms)	Power (W)	Energy pr. iter. (mJ)	Time pr. iter. (ms)	Power (W)	Energy pr. iter. (mJ)
Dense Directed	0.019	3.32	0.064	0.477	0.488	0.232
Dense Undirected	0.021	3.377	0.072	0.488	0.505	0.246
Sparse Directed	0.015	3.242	0.049	0.483	0.448	0.216
Sparce Undirected	0.015	3.260	0.049	0.49	0.446	0.218

6.1. Algorithm Characteristics and Hardware Suitability

Heapsort and Dijkstra's algorithms have different levels of inherent parallelism and complexity. Heapsort may benefit more from native parallelization in hardware due to its ability to efficiently parallelize the comparison and swapping elements, especially when sorting larger datasets. In contrast, our implementation of Dijkstra's algorithm may not fully leverage FPGA capabilities due to its sequential dependencies, leading to less impressive gains or even inefficiencies on FPGAs.

While our results are less promising for Dijkstra, previous studies show that FPGA implementations of Dijkstra can provide considerable performance optimization compared to software due to their different growth rates: "The average execution time of the FPGA-based version grew only linearly, whereas the average execution time of the microprocessor-based version displayed quadratic growth" [28], see Table 5. Thus, it may be that increasing the graph size can improve the results, but we speculate that our result is a consequence of our implementation. Future work would include reimplementations and energy evaluations of previously successful hardware implementations.

Table 5. Dijkstra performance improvements by Tommiska et al. [28] when using an FPGA solution compared to a Microprocessor solution

Vertices	**Logic Elements**	**Memory**	**FPGA time**	**Microprocessor time**	**Speedup**
8	834	632 bits	10.6µs	250µs	23.58
16	1536	2116 bits	13.4µs	434µs	32.39
32	2744	8287 bits	17.2µs	802µs	46.63
64	5100	32894 bits	21.6µs	1456µs	67.41

6.2. Energy Efficiency versus Performance

Both algorithms emphasize the trade-offs between performance gains and energy efficiency. In both algorithms, the FPGAs increased execution time. Energy and time are highly correlated. However, comparing the execution time across software and hardware does not necessarily indicate a similar pattern for their energy consumption.

The benefit of the FPGA is that the power dissipation is 5 to 8 times lower than the Raspberry Pi's power dissipation. Therefore, a good rule of thumb would be that FPGA is a good choice when the execution time of the FPGA is 5 to 8 times faster than the Raspberry Pi. In our study, this is the case for Heapsort but not for Dijkstra. In addition, this factor may change with a different choice of hardware. Future work would include evaluating the energy and time trade-off factors for various FPGAs and computers.

6.3. System Architecture Design

Insights from both algorithms can guide system architects in designing more efficient systems by choosing the right combination of hardware and software based on the specific algorithms they expect to run most frequently. There is no clear case for always employing FPGAs to provide energy reductions equivalently to performance. However, in some cases, the developer may be able to exploit native parallelism and ensure that hardware implementation has a slower execution time growth rate for increasing input sizes. It would be beneficial to evaluate the energy consumption of high-performing hardware implementations and to identify trade-off trends on the architectural design level.

6.4. Hypotheses on Algorithmic Performance

Our results demonstrate that Heapsort, with its potential for parallel processing, capitalized on the FPGA's architecture to yield significant energy savings. One hypothesis for this outcome is that the FPGA's ability to conduct multiple comparisons in parallel, particularly through the efficient use of LUTs and parallel memory access, minimizes both time and energy when compared to the CPU's more serial processing. For instance, as the input size increased, the FPGA continued to scale effectively, likely due to its architectural suitability for handling concurrent operations. This suggests that algorithms with similar local parallelism would exhibit comparable performance benefits.

In contrast, Dijkstra's algorithm, with its inherent sequential dependencies, struggled to take advantage of FPGA's strengths. We hypothesize that this inefficiency stems from the algorithm's need to update path lengths iteratively, which bottlenecks performance and limits potential energy gains. Future research could investigate whether alternate graph traversal algorithms with fewer sequential dependencies, or different graph configurations (such as sparse versus dense graphs), might better exploit hardware acceleration. Additionally, hardware design optimizations targeting these sequential steps could mitigate some of these limitations.

7. The Limits of Speedup

Building on the hypothesis that algorithms with local parallelism would exhibit similar performance benefits, we explored the upper bounds of FPGA speedup using the highly parallelizable Conway's Game of Life [4]. This experiment serves to demonstrate how extreme parallelism can push the limits of performance and energy efficiency on FPGA, further supporting the idea that parallel algorithms are well-suited for hardware acceleration. Conway's Game of Life is a zero-player game defined on cellular automata. The cellular automata are 2D grids called worlds, where each grid cell has eight neighbors, and each cell can have one of two states: dead or alive. For each step, the cell states are updated according to their state and the states of their neighboring cells in the previous step.

1. Any live cell with two or three live neighbors survives.
2. Any dead cell with three live neighbors becomes a live cell.
3. All other live cells die in the next generation. Similarly, all other dead cells stay dead.

The initial state is given as input to the program. Because each cell depends only on nearby cells, Game of Life is highly parallelizable.

Table 6. The execution time in us and speed-up of FPGA over software executions

		Execution time per step (us)			**FPGA Speedup**	
World	**Cells**	Mac	Raspberry Pi	FPGA	Mac	Raspberry Pi
10x10	100	0.10	1.783	0.0040	25	445
20x20	400	0.33	5.137	0.0040	82	1284
30x30	900	0.70	9.965	0.0041	170	2430
40x40	1600	1.21	17.212	0.0040	302	4302
50x50	2500	1.81	25.204	0.0044	411	5728
60x60	3600	2.76	37.822	0.0045	613	8404
70x70	4900	3.54	57.665	0.0040	884	14416
80x80	6400	4.81	64.396	0.0047	1023	13701
90x90	8100	6.50	81.309	0.0045	1444	18068
100x100	10000	7.51	109.964	0.0048	1564	22909

Table 6 shows the average execution time for a single time step and the speedup provided by the FPGA implementation compared to the Java software implementation executed on a MacBook Pro and the Raspberry Pi. These results show that the gain in performance increases with the measured world sizes.

Speedup factors range from 25 for a 10x10 world to 1500 for a 100x100 world. The speedup scales linearly with the problem size. While the Game of Life is an artificial workload, its parallelizable nature made it an ideal candidate for indicating an upper bound for speedups when moving algorithms from software into an FPGA.

7.1. Resource Utilization

We have implemented the Game of Life of different sizes in a Cyclon IV FPGA found on the DE2-115 evaluation board. We report the design size in logic elements and registers. A logic element represents one 4-bit lookup table. For synthesis, we used the Quartus 19.1.0 Lite Edition.

Table 7 shows the FPGA implementation's resource consumption for different world sizes. We can see that the size grows linear. The maximum frequency of the circuit is reported between 209 MHz and 250 MHz. Therefore, when we assume running it at 200 MHz we can compute one iteration in 5 ns.

Table 7. The resource utilization and minimum iteration time of different sized Game of Life worlds in an FPGA

World	Logic Elements	Registers	Minimum Clock Period
10 x 10	804	104	4.0 ns
20 x 20	3539	404	4.0 ns
30 x 30	7995	904	4.1 ns
40 x 40	14463	1604	4.0 ns
50 x 50	23439	2504	4.4 ns
60 x 60	34414	3604	4.5 ns
70 x 70	45119	4904	4.0 ns
80 x 80	59136	6404	4.7 ns
90 x 90	75102	8104	4.5 ns
100 x 100	97871	10004	4.8 ns

As expected, we use one register per cell. However, the number of logic elements per cell is surprisingly high, an average of around 9 logic elements per cell. We assume that the Chisel PopCount method has some room for improvement. However, as we aim for a technique that enables software developers to describe their algorithms in hardware, we are avoiding optimization tricks.

7.2. Estimated Energy Consumption

We did not measure power or energy consumption of the FPGA implementation. However, the DE2-115 FPGA board comes with a power supply of 24 W. Therefore, this is the upper bound of power consumption of the whole FPGA board, including peripheral devices and external memories.

If we assume 24 W as an upper bound on the power consumption and an operating frequency of 200MHz, then one iteration of a 100 x 100 Game of Life world consumes 96nJ. In comparison, the Raspberry Pi has been reported to consume an average of 6.4 W when all four cores are busy[6] and one iteration of a 100x100 world takes 0.109964 ms. Thus, a conservative energy consumption estimate for one iteration of a 100x100 world is 0.703769 mJ. From these conservative estimates, the hardware implementation can significantly improve energy consumption compared to the Raspberry Pi 4.

8. Related Work

FPGA research spans a diverse range of applications, including machine learning, signal processing, communication systems, and big data [25]. Sorting algorithms, while less commonly studied, represent a valuable niche for benchmarking hardware capabilities due to their computational diversity and potential for parallelism. There are numerous studies on hardware implementations of algorithms (e.g., [7, 12, 14, 15, 19, 20, 22, 28, 30]); however, only a few focus on energy consumptionor compare performance across platforms.This work fills these gaps by focusing on energy and runtime variability across

[6] `https://www.pidramble.com/wiki/benchmarks/power-consumption`

multiple platforms, emphasizing the reproducibility of evaluations and the role of algorithm characteristics.

Jmaa et al. [14] study the implementation of sorting algorithms on FPGAs using high-level synthesis, comparing their performance with CPU-based implementations. They evaluate algorithms such as BubbleSort and QuickSort, focusing on execution time and its variability to demonstrate the acceleration benefits of FPGAs. Their results show that FPGAs significantly outperform CPUs in execution time for sorting tasks, particularly for larger input sizes. Their results align with ours in highlighting the benefits of FPGAs for parallelizable tasks like Heapsort, where execution time improves significantly compared to CPUs. However, their study does not consider energy efficiency, a key focus of our work, nor does it explore intra-platform variability or reproducibility. However, their work does not explore the impact of algorithm characteristics, such as parallelism, on energy efficiency, a key focus of our study.

In a separate study, Jmaa et al. [15] analyze sorting algorithms implemented as software on ARM Cortex A9 processors, part of the Zynq platform. They measure computational time, energy consumption, and stability to determine the most efficient sorting algorithm for embedded systems. Their results highlight ShellSort as the most efficient algorithm for larger input sizes, with a high correlation between energy consumption and execution time. This study is close in content to our study, but they do not compare the hardware implementations with other implementations. Instead, they compare the time and energy usage of their FPGA implementations. Their results partially align with ours by showing that energy consumption is highly correlated with execution time.

Asiatici et al. [2] investigate the use of FPGAs as accelerators in a hybrid CPU-FPGA system for parallel string sorting. Their results demonstrate that FPGAs can significantly reduce runtime and improve energy efficiency for classification tasks, complementing CPU operations. While their study highlights FPGA efficiency and energy efficiency for parallel workloads, its focus on hybrid systems differs from our evaluation of standalone FPGA and CPU performance.

Mihhailov et al. [21] propose parallel FPGA-based implementations of recursive sorting algorithms, leveraging hierarchical finite state machines to improve performance. They focus on execution time, demonstrating significant performance gains through parallelization on FPGAs. Their results align with ours in showing the benefits of FPGAs for parallelizable tasks like Heapsort, particularly in terms of execution time. However, their study does not include energy efficiency analysis or comparisons with CPUs.

Zhou et al. [30] investigate the implementation of the A* algorithm on FPGAs for real-time path planning, leveraging a custom hardware architecture to optimize performance. Their study primarily focuses on execution time and resource utilization, demonstrating significant speed improvements over CPU implementations, particularly for large-scale pathfinding tasks. Their results align with ours in showing that FPGAs can speed up computationally intensive tasks but differ in focusing exclusively on path planning rather than broader algorithmic comparisons. Unlike our study, Zhou et al. do not consider energy efficiency or intra-platform variability.

Koch et al. [19] explore FPGA-based sorting architectures designed for large datasets, leveraging run-time reconfiguration to optimize performance. They measure execution time and demonstrate the scalability of their approach, particularly for sorting tasks that require high throughput. Their results align with ours in showcasing the benefits of FPGAs for sorting tasks like Heapsort, particularly for large input sizes. However, they do not address energy consumption, a central focus of our study.

Mueller et al. [22] investigate data processing on FPGAs, comparing their performance with modern CPUs. They focus on asynchronous sorting networks and their integration into heterogeneous computing systems. Their results indicate that FPGAs can achieve competitive performance with CPUs while offering significant advantages in power consumption. Their study aligns with ours in comparing CPUs and FPGAs but differs in its broader focus on data processing and integration strategies rather than algorithm-specific energy and runtime variability.

Tommiska and Skyttä [28] implement Dijkstra's shortest path algorithm on FPGAs, achieving notable speed advantages over traditional CPU-based implementations. Their study focuses on execution time improvements enabled by FPGA-specific optimizations, demonstrating the potential for hardware acceleration in pathfinding tasks. As discussed earlier in Section 6, their performance results do not fully align with ours. They find FPGAs to be advantageous for Dijkstra's algorithm and we speculate that result difference is due to our hardware implementation. Their study does not explore energy efficiency.

Lei et al. [20] propose an FPGA implementation of the single-source shortest path (SSSP) problem using a systolic array priority queue. They evaluate the performance of their approach in terms of execution time and scalability. Their results show that FPGAs outperform CPUs for large-scale graph problems. Their results partially align with ours, as they highlight the advantages of FPGAs for large-scale, parallelizable problems, but differs in evaluating only execution time without considering energy efficiency.

Favaro et al. [7] compare multi-core CPUs and FPGAs for numerical linear algebra tasks, such as dense matrix multiplication and sparse matrix-vector multiplication. Their findings show that multi-core CPUs outperform FPGAs in runtime for smaller or less parallelizable tasks, benefitting from efficient caching and optimized libraries like Intel MKL. In contrast, FPGAs excel in energy efficiency for larger, highly parallelizable workloads like SPMV, though their runtime advantage remains limited. This aligns with our findings for tasks like Heapsort, which leverage parallelism effectively on FPGAs, and contrasts with sequential tasks like Dijkstra's algorithm, where CPUs dominate. While their study aligns with our emphasis on task-dependent hardware suitability, it does not address how software variability or task design impacts energy and runtime efficiency within a single hardware platform, which is central to our work.

9. Conclusion

This study compared the energy efficiency and performance of software and hardware implementations of Heapsort and Dijkstra's algorithms. Our findings reveal both the advantages and limitations of using FPGAs compared to traditional software implementations on a Raspberry Pi.

For Heapsort, the hardware implementation on an FPGA demonstrated a clear advantage in terms of energy efficiency, confirming the potential of FPGAs for algorithms where some operations can be done in parallel, The results showed that the energy consumption for Heapsort on FPGA was consistently lower than on the Raspberry Pi, particularly as input sizes increased. This suggests that FPGAs can effectively reduce energy consumption for tasks where some parallel processing can be exploited.

In contrast, Dijkstra's algorithm did not exhibit the same level of energy efficiency on FPGA. Despite FPGAs' inherent capabilities for handling parallel tasks, the complex dependencies and sequential nature of Dijkstra's algorithm limited the expected gains. The study highlighted that traditional CPU implementations might still hold an advantage in terms of both performance and energy consumption for algorithms with significant sequential operations.

The comparison also underscored the importance of choosing hardware or software solutions based on the specific requirements and characteristics of the algorithm. While FPGAs offer considerable reductions in power dissipation, they are not universally superior for all computational tasks. Our findings suggest that the decision to use FPGA over CPU should be guided by a more detailed knowledge of the algorithm's structure and the potential for parallelism.

Additionally, this study contributes to the ongoing discussion about the trade-offs between computational speed and energy efficiency. It provides a first step towards a nuanced perspective that can aid system architects and developers in making informed decisions about the hardware-software configurations that best meet their performance and efficiency goals.

While this study focuses on Heapsort and Dijkstra's algorithm, both of which offer limited parallelism, future work will explore more parallelizable algorithms, such as quicksort, matrix operations or image processing, to better demonstrate FPGA's energy and performance potential. Our preliminary results with Conway's Game of Life (Section 7) show that substantial speedups can be achieved through parallel computation, and similar gains are expected from a wider range of algorithms. This suggests that algorithms with similar characteristics can significantly reduce energy consumption by distributing computational tasks evenly across FPGA's processing elements. We hypothesize that applying this approach to a wider range of highly parallelizable algorithms will result in similar improvements in both energy efficiency and performance, as distributing tasks across FPGA's processing elements can significantly reduce energy consumption. These extensions will provide further insights into how parallelism can be optimized to enhance both computational speed and energy savings across a variety of workloads.

Future work will focus on optimizing hardware implementations and expanding the range of algorithms tested to further explore their energy and performance potential. Additionally, we plan to evaluate the scalability and applicability of our findings across a wider variety of hardware platforms, including more powerful multicore processors, GPUs, and different FPGA models and configurations. This broader assessment will help establish more generalized guidelines for selecting between software and hardware implementations, particularly in terms of energy efficiency and performance metrics.

References

1. Andrae, A.S.G.: New perspectives on internet electricity use in 2030. Engineering and Applied Science Letter 3, 19–31 (2020), https://doi.org/10.30538/psrp-easl2020.0038
2. Asiatici, M., Maiorano, D., Ienne, P.: How many cpu cores is an fpga worth? lessons learned from accelerating string sorting on a cpu-fpga system. Journal of Signal Processing Systems 93, 1405–1417 (12 2021), https://doi.org/10.1007/s11265-021-01686-8

3. Bachrach, J., Vo, H., Richards, B.C., Lee, Y., Waterman, A., Avizienis, R., Wawrzynek, J., Asanovic, K.: Chisel: constructing hardware in a scala embedded language. In: Groeneveld, P., Sciuto, D., Hassoun, S. (eds.) The 49th Annual Design Automation Conference 2012, DAC '12, San Francisco, CA, USA, June 3-7, 2012. pp. 1216–1225. ACM (2012), https://doi.org/10.1145/2228360.2228584
4. Berlekamp, E.R., Conway, J.H., Guy, R.K.: Winning ways for your mathematical plays. Vol. 2. Academic Press Inc., London (1982), games in particular
5. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. MIT Press, Cambridge, MA, 3rd edn. (2009)
6. Eder, K., Gallagher, J.P., López-García, P., Muller, H., Banković, Z., Georgiou, K., Haemmerlé, R., Hermenegildo, M.V., Kafle, B., Kerrison, S., Kirkeby, M., Klemen, M., Li, X., Liqat, U., Morse, J., Rhiger, M., Rosendahl, M.: Entra: Whole-systems energy transparency. Microprocessors and Microsystems 47, 278–286 (2016), https://doi.org/10.1016/j.micpro.2016.07.003
7. Favaro, F., Dufrechou, E., Ezzatti, P., Oliver, J.P.: Energy-efficient algebra kernels in fpga for high performance computing. Journal of Computer Science & Technology 21(2), 80–92 (2021)
8. Field, H., Anderson, G., Eder, K.: Eacof: a framework for providing energy transparency to enable energy-aware software development. In: Proceedings of the 29th Annual ACM Symposium on Applied Computing. p. 1194–1199. SAC '14, Association for Computing Machinery, New York, NY, USA (2014), https://doi.org/10.1145/2554850.2554920
9. Hähnel, M., Döbel, B., Völp, M., Härtig, H.: Measuring energy consumption for short code paths using rapl. SIGMETRICS Perform. Eval. Rev. 40(3), 13–17 (Jan 2012), https://doi.org/10.1145/2425248.2425252
10. Hirata, H., Nunome, A.: A modified parallel heapsort algorithm. International Journal of Software Innovation 8(3), 1–18 (2020), https://doi.org/10.4018/IJSI.2020070101
11. Intel Corporation: Intel 64 and IA-32 Architectures Software Developer's Manual (2023), https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html, accessed: 2024-11-04
12. Jagadeesh, G., Srikanthan, T., Lim, C.: Field programmable gate array-based acceleration of shortest-path computation. IET Computers & Digital Techniques 5, 231–237 (2011), https://doi.org/10.1049/iet-cdt.2009.0072
13. Jasika, N., Alispahic, N., Elma, A., Ilvana, K., Elma, L., Nosovic, N.: Dijkstra's shortest path algorithm serial and parallel execution performance analysis. In: 2012 Proceedings of the 35th International Convention MIPRO. pp. 1811–1815 (2012), https://ieeexplore.ieee.org/document/6240942, accessed: 2024-11-04
14. Jmaa, Y.B., Atitallah, R.B., Duvivier, D., Jemaa, M.B.: A comparative study of sorting algorithms with fpga acceleration by high level synthesis. Computacion y Sistemas 23, 213–230 (2019), https://doi.org/10.13053/CyS-23-1-2999
15. Jmaa, Y.B., Duvivier, D., Abid, M.: Sorting algorithms on arm cortex a9 processor. In: Barolli, L., Woungang, I., Enokido, T. (eds.) International Conference on Advanced Information Networking and Applications. vol. 227, pp. 355–366. Springer International Publishing (2021), https://doi.org/10.1007/978-3-030-75078-7_36
16. Khan, K.N., Hirki, M., Niemi, T., Nurminen, J.K., Ou, Z.: RAPL in Action. ACM Transactions on Modeling and Performance Evaluation of Computing Systems 3(2), 1–26 (jun 2018), https://doi.org/10.1145/3177754
17. Kirkeby, M.H., Krabben, T., Larsen, M., Mikkelsen, M.B., Petersen, T., Rosendahl, M., Schoeberl, M., Sundman, M.: Energy consumption and performance of heapsort in hardware and software (2022), https://arxiv.org/abs/2204.03401
18. Kirkeby, M.H., Schoeberl, M.: Towards comparing performance of algorithms in hardware and software (2022), https://arxiv.org/abs/2204.03394

19. Koch, D., Torresen, J.: Fpgasort: a high performance sorting architecture exploiting run-time reconfiguration on fpgas for large problem sorting. In: Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays. p. 45–54. FPGA '11, Association for Computing Machinery, New York, NY, USA (2011), `https://doi.org/10.1145/1950413.1950427`
20. Lei, G., Dou, Y., Li, R., Xia, F.: An fpga implementation for solving the large single-source-shortest-path problem. IEEE Transactions on Circuits and Systems II: Express Briefs 63(5), 473–477 (2016), `https://doi.org/10.1109/TCSII.2015.2505998`
21. Mihhailov, D., Sklyarov, V., Skliarova, I., Sudnitson, A.: Parallel fpga-based implementation of recursive sorting algorithms. In: 2010 International Conference on Reconfigurable Computing and FPGAs. pp. 121–126 (2010)
22. Mueller, R., Teubner, J., Alonso, G.: Data processing on fpgas. Proc. VLDB Endow. 2(1), 910–921 (aug 2009), `https://doi.org/10.14778/1687627.1687730`
23. Pereira, R., Couto, M., Ribeiro, F., Rua, R., Cunha, J., Fernandes, J.P., Saraiva, J.: Ranking programming languages by energy efficiency. Sci. Comput. Program. 205, 102609 (2021), `https://doi.org/10.1016/j.scico.2021.102609`
24. Rotem, E., Naveh, A., Ananthakrishnan, A., Weissmann, E., Rajwan, D.: Power-management architecture of the intel microarchitecture code-named sandy bridge. IEEE Micro 32(2), 20–27 (2012), `https://doi.org/10.1109/MM.2012.12`
25. Ruiz-Rosero, J., Ramirez-Gonzalez, G., Khanna, R.: Field programmable gate array applications—a scientometric review. Computation 7(4) (2019), `https://www.mdpi.com/2079-3197/7/4/63`
26. Santos, B., Fernandes, J.P., Kirkeby, M.H., Pardo, A.: Compiling haskell for energy efficiency: Empirical analysis of individual transformations. In: The 39th ACM/SIGAPP Symposium on Applied Computing (SAC '24), April 8–12, 2024, Avila, Spain. Association for Computing Machinery, New York, NY, United States (2023), `https://doi.org/10.1145/3605098.3635915`
27. Schoeberl, M.: Digital Design with Chisel. Kindle Direct Publishing (2019), available at `https://github.com/schoeberl/chisel-book`
28. Tommiska, M., Skyttä, J.: Dijkstra's shortest path routing algorithm in reconfigurable hardware. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 2147, 653–657 (2001), `https://doi.org/10.1007/3-540-44687-7_73`
29. Xilinx: Vivado design suite user. guide power analysis and optimization (October 2021), `https://docs.amd.com/r/2021.2-English/ug907-vivado-power-analysis-optimization`, uG907 (v2021.2), Accessed: 2024-11-04
30. Zhou, Y., Jin, X., Wang, T.: FPGA implementation of a$_*$ algorithm for real-time path planning. Int. J. Reconfigurable Comput. 2020, 8896386:1–8896386:11 (2020), `https://doi.org/10.1155/2020/8896386`

Maja H. Kirkeby is Associate Professor at Roskilde University, Denmark, specializing in energy consumption of software and IT systems.

Thomas Krabben is Lab Manager at FlexLab, Roskilde University, Denmark, with expertise in hardware and IT systems setup and operational management.

Mathias Larsen is MSc student at IT University, Denmark, with focus on in energy consumption of software algorithms.

Maria B. Mikkelsen is Research Assistant at Roskilde University, Denmark, specialized in energy consumption and program transformation ofcomputational systems.

Mads Rosendahl is Associate Professor at Roskilde University, Denmark, with expertise in program analysis and program transformation.

Martin Sundman is MSc student at IT University, Denmark, specialized in energy consumption of software algorithms.

Tjark Petersen is MSc student at DTU Compute, Denmark, expertise in efficient hardware algorithm design and implementation.

Martin Schoeberl is Professor at DTU Compute, Denmark, world-leading research on real-time systems and hardware design.

Received: September 14, 2024; Accepted: December 19, 2024.