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

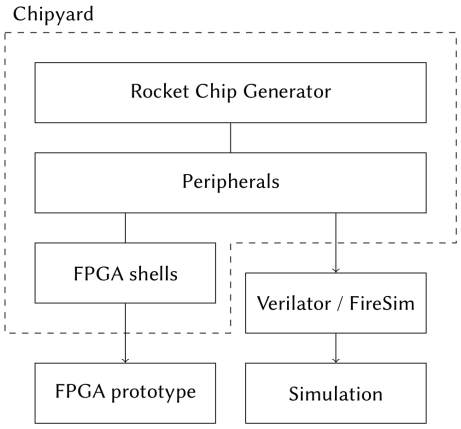

Fig. 1. Overview of generators of Chipyard [2]. The resulting code can be synthesised for an FPGA or simulated

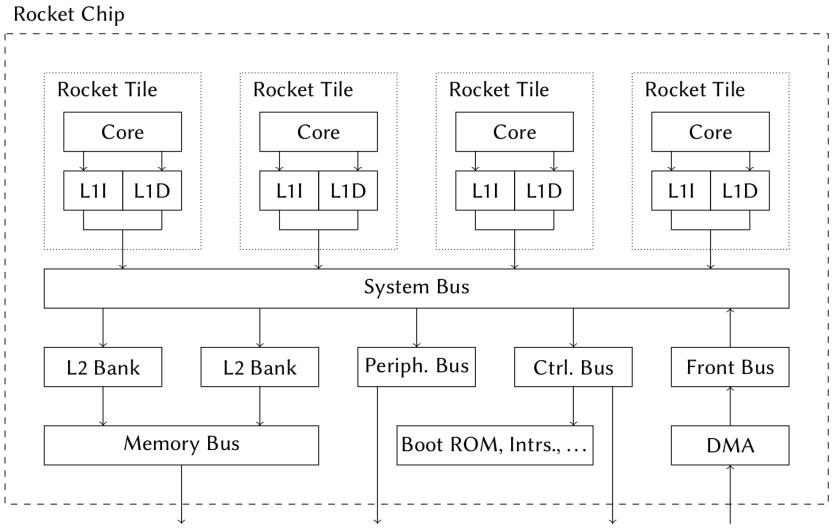

Fig. 2. The Rocket Chip [5] consists of multiple processor tiles, and devices connected through dedicated buses, like memory and peripherals. All parts of the chip communicate through the system bus

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 fuflfill 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

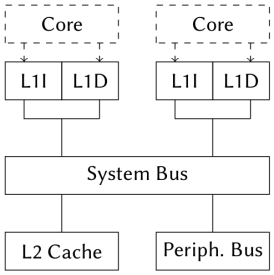

Fig. 3. Elements of interest to evaluate interferences in the memory hierarchy: private L1 caches, shared L2 caches, buses that connect shared resources, and the shared system bus itself

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

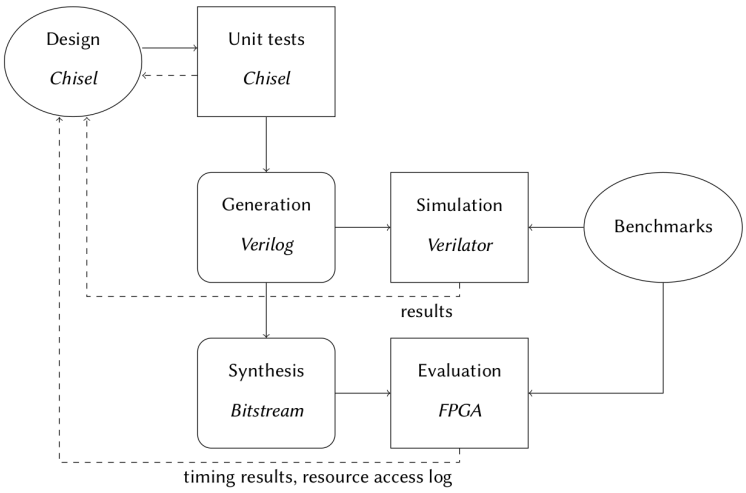

Fig. 4. Overview of the prototyping flow with Manatee. Results of the unit tests provide immediate feedback, which can be further tested in the Verilator simulation. The evaluation of the bitstream on the FPGA provides accurate timing results and logs of the resource accesses

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-

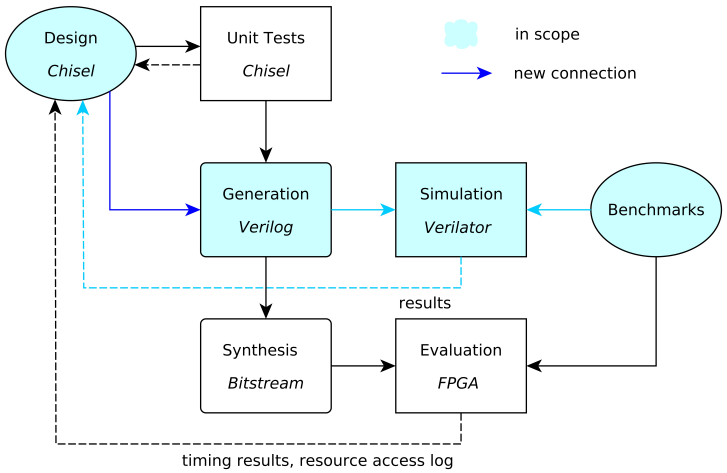

Fig. 5. The concept with a marked scope of this extension

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.

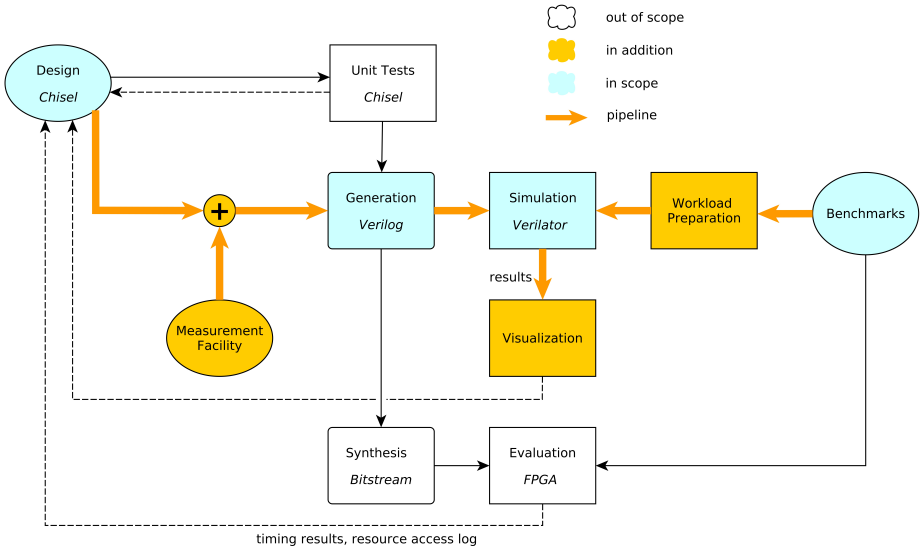

Fig. 6. Intermediate steps to the conceptual prototyping flow from Fig. 4

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)

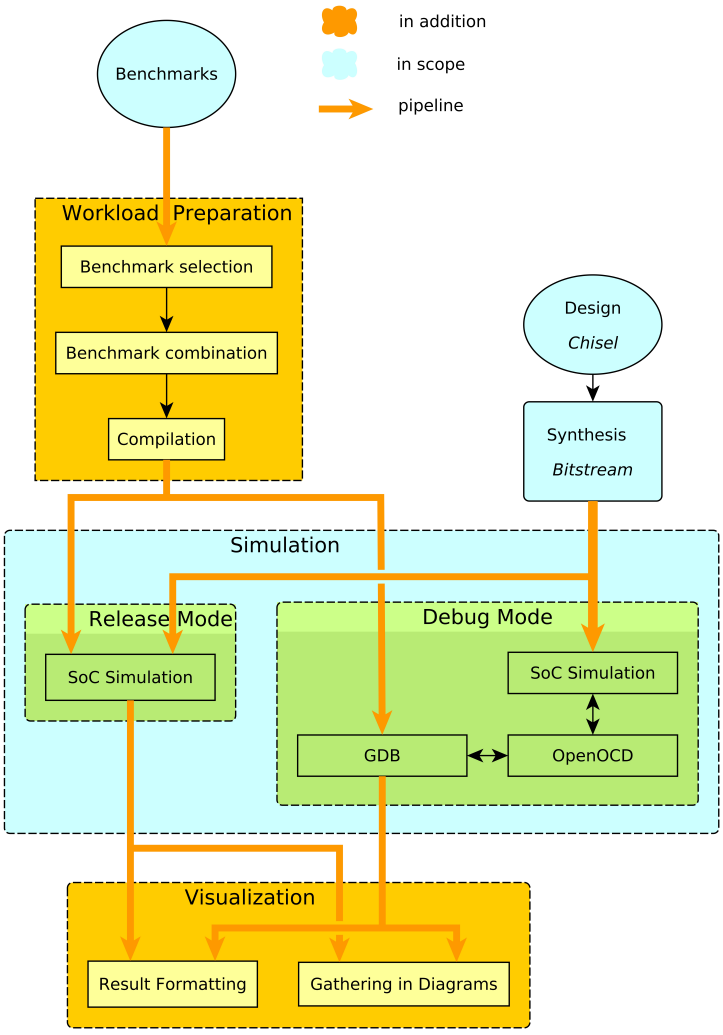

Fig. 7. An overview of the main pipeline parts and their connections

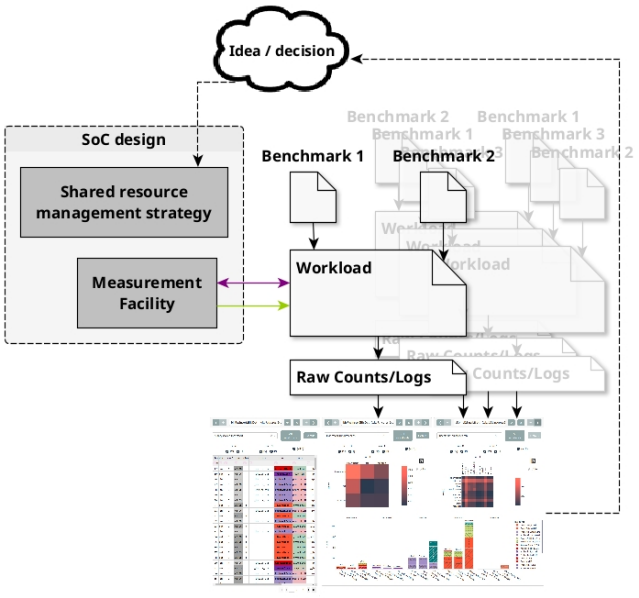

Fig. 8. The prototyping flow from a data-focused perspective

- 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.

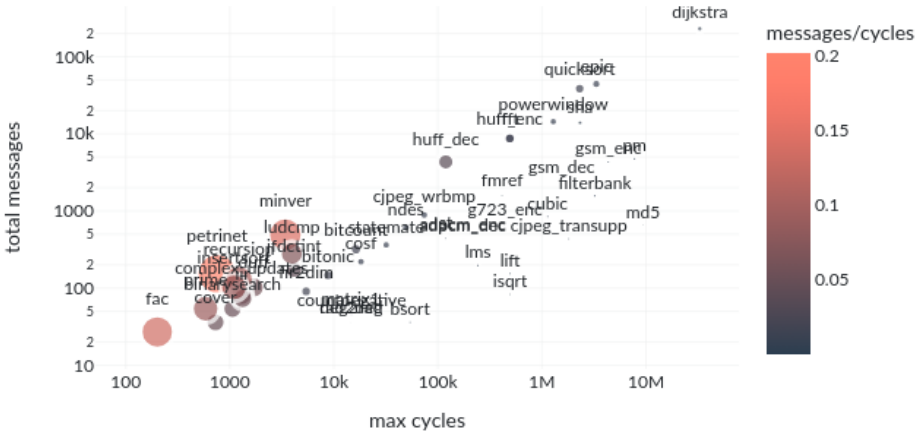

Fig. 9. A graph produced by Manatee showing the number of messages exchanged between the L2 cache and the L1 data and instruction caches over execution time for single benchmark workloads. Log mode is activated to better fit all workloads

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

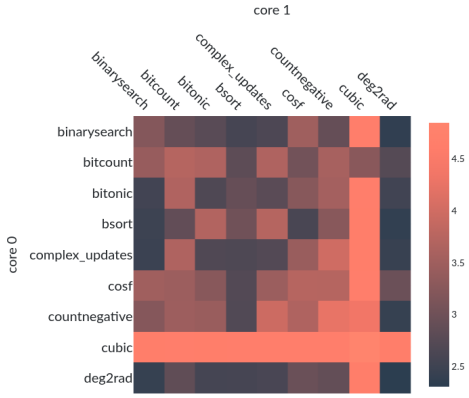

Fig. 11. The same visualisation setting and similar hardware. Here each core has 32 L1 data cache sets

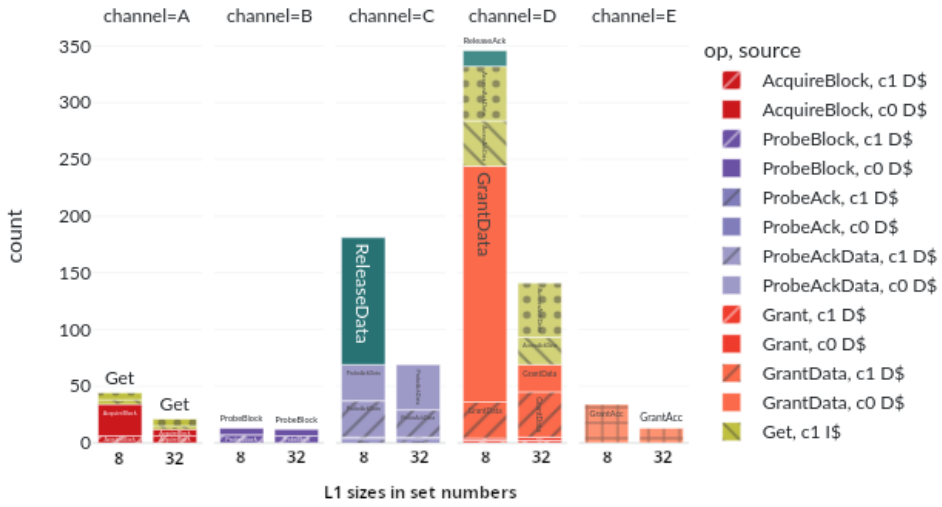

Fig. 12. Comparing the L2 cache accesses between two different hardware configurations for countnegative on the first and deg2rad on the second core

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 ⟶ Grant(very small)/GrantData ⟶ GrantAck
- Green colours: ReleaseData ⟶ ReleaseAck
- Purple colours: ProbeBlock ⟶ 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

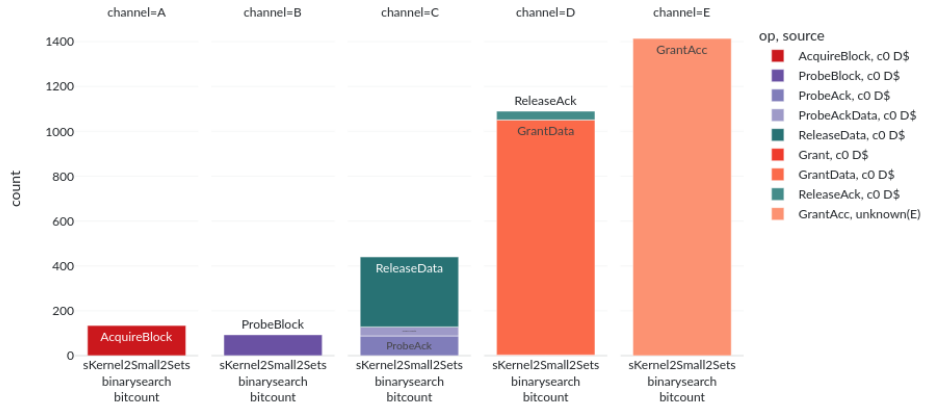

Fig. 13. Hardware: Two small Rocket cores, each with only 2 sets in their L1 caches; Workload: core 1: binarysearch, core 2: bitcount; The graph shows messages between L2 cache and the L1 data cache of the first core

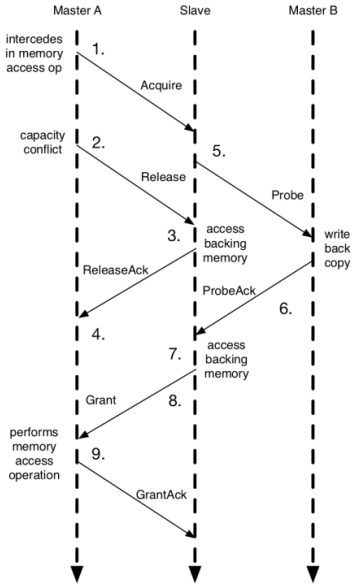

Fig. 14. Taken from [35, 73]. This graph shows one of the most common interactions between TL-C level message flows. In our case the masters are L1 caches and the slave is the L2 cache

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

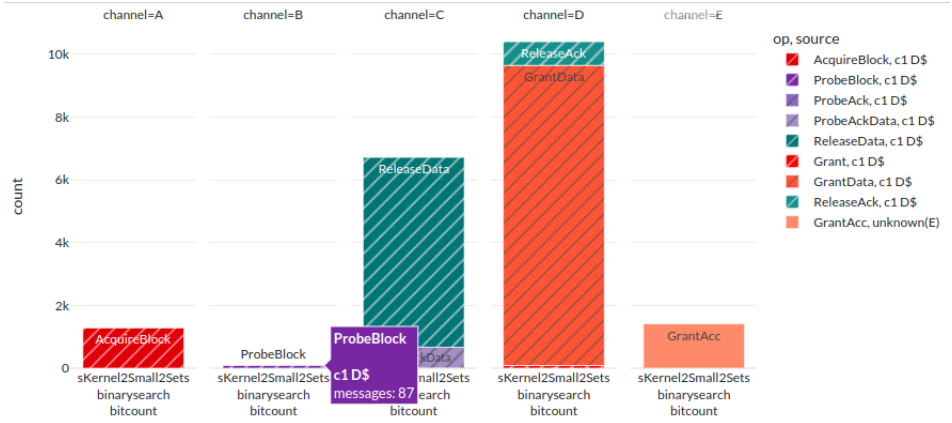

Fig. 15. Same data set as Fig. 13; In contrast, this figure shows the counts of bitcount executed on the second core

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.

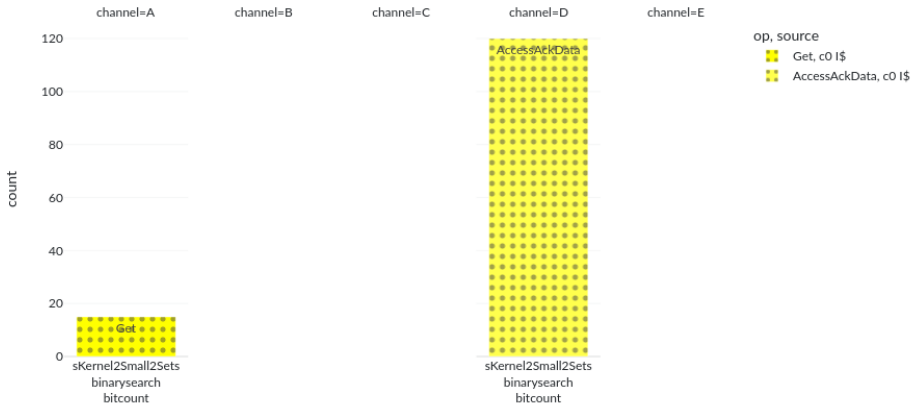

Fig. 16. Same data set as Fig. 13 and 15; Here instead of the counts of an L1 data cache L1 instruction cache counts are selected

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

```
4095
4096  00000000800024b8 <_global_atexit>:
4097      ...
4098
4099  00000000800024c0 <binarysearch_data>:
4100      ...
4101
4102  0000000080002538 <insertsort_a>:
4103      ...
4104
4105  0000000080002564 <__boot_sync>:
4106      80002564:   0000                    unimp
4107      ...
```

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.

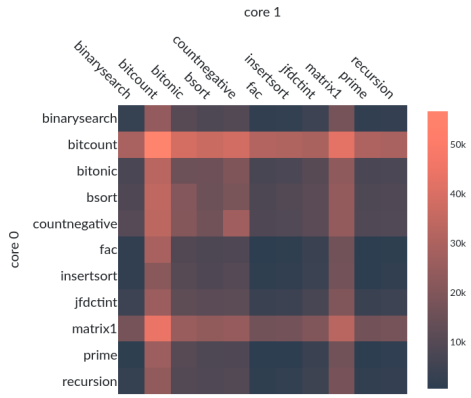

Fig. 20. Hardware: Two Small Rocket cores, each has L1 caches with only two sets. The data is from a selection of kernel benchmarks

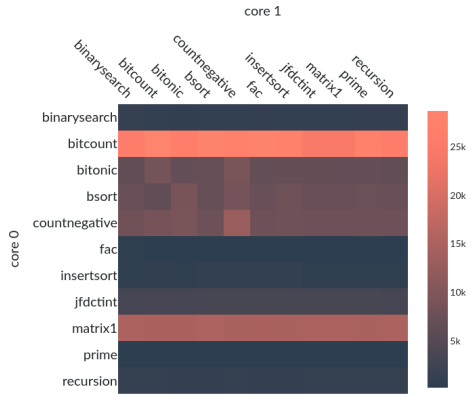

Fig. 21. Setup is the same as in Fig. 20, but only counts of messages from and to the first core (core 0) are shown

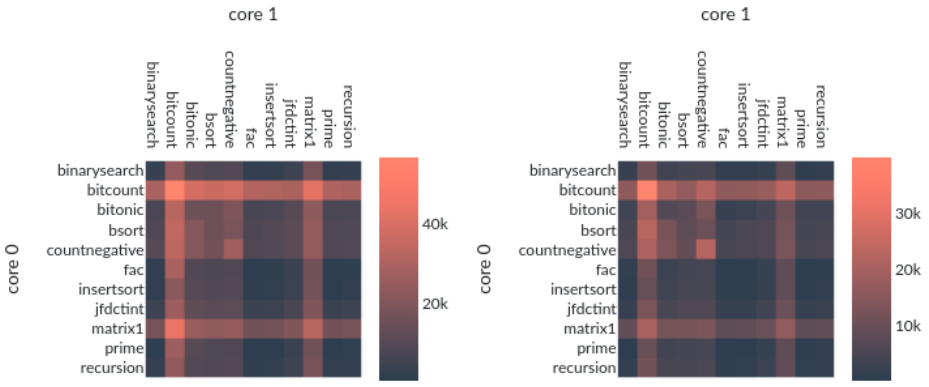

Fig. 22. Left setup is the same as in Fig. 20. The right setup has double the amount of cache sets (Notice the different heatmap scales on the right of each graph)

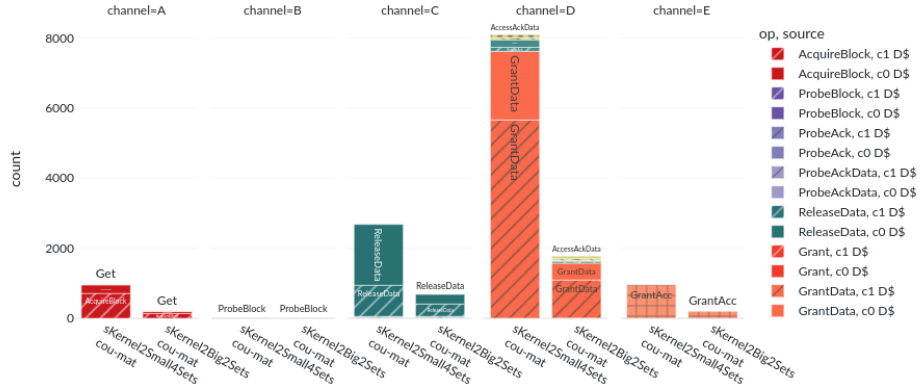

Fig. 23. Here, the visualiser pairs the message counts of different hardware configurations per channel to facilitate their comparison. The left setup is the same as the improvement of the previous iteration. The right setup has big Rocket cores but only half the number of cache sets

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.

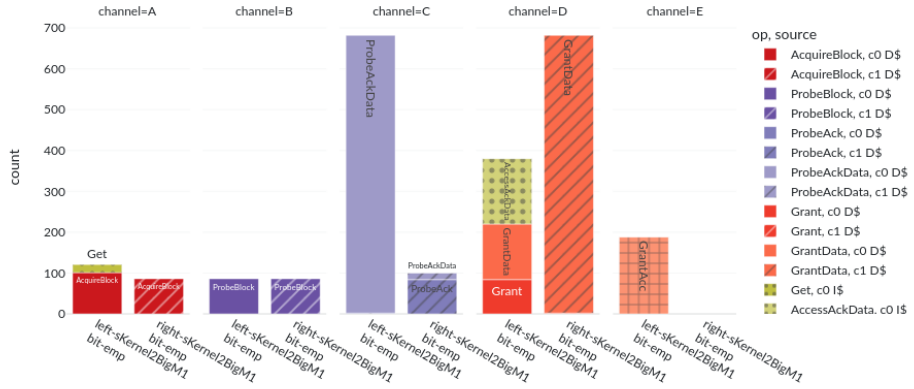

Fig. 24. Comparing the L2 cache accesses between two different cores. The workload consists of the bitcount benchmark running on the first core. The second core does not have a task assigned

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

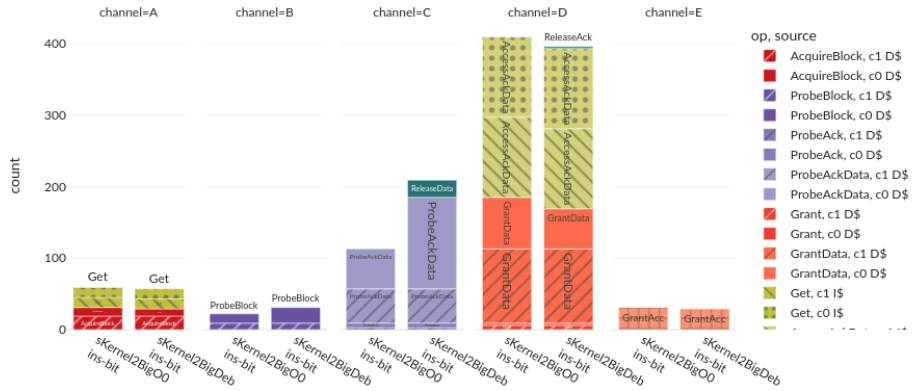

Fig. 25. Both sides use the same hardware simulator and the same workload code. However, the right side was executed in debug mode while the left side was executed regularly

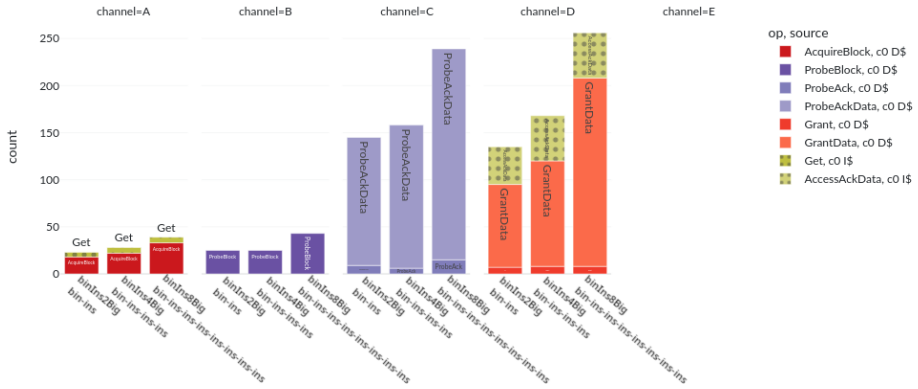

Fig. 26. The left bars of each channel's section represent dual-core workloads, the middle quad-core, and the right bars octo-cores. On the first core runs binarysearch while the tasks of the remaining cores consist of insertsorts. Each bar shows only messages associated with the first core

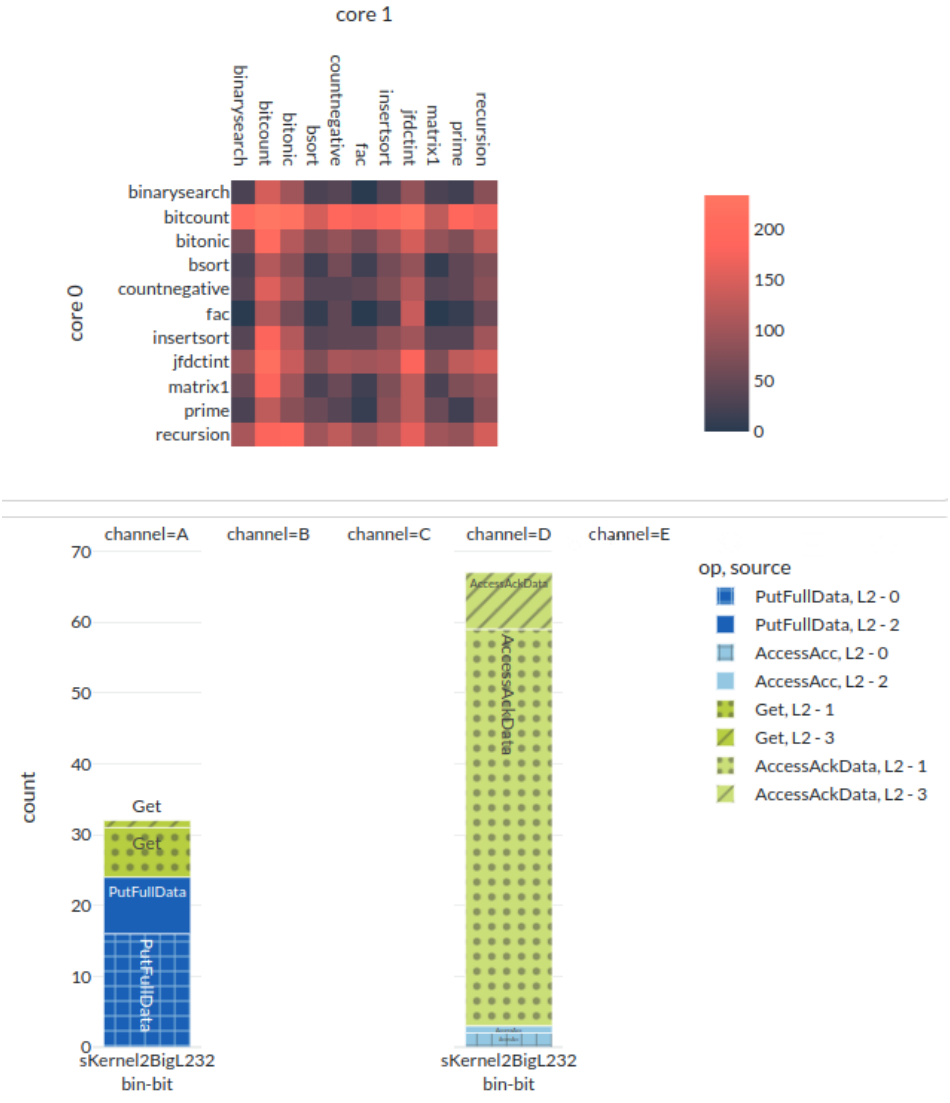

Fig. 27. On the top, a heatmap shows the message counts of workloads made up of a selection of kernel benchmarks. The lower part depicts a bar chart of the workload binarysearch-bitcount. Here the measurement facility monitored the L2 - memory bus connection instead of the system bus - L2 connection

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.