UDC 004.31.26, DOI: 10.2298/CSIS1001189W

The Design and Evaluation of Hierarchical Multi-level Parallelisms for H.264 Encoder on Multi-core Architecture

Haitao Wei[1], Junqing Yu[1], and Jiang Li[1]

[1]School of Computer Science & Technology,
Huazhong University of Science & Technology, 430074 Wuhan, China
yjqing@hust.edu.cn

Abstract. As a video coding standard, H.264 achieves high compress rate while keeping good fidelity. But it requires more intensive computation than before to get such high coding performance. A Hierarchical Multi-level Parallelisms (HMLP) framework for H.264 encoder is proposed which integrates four level parallelisms – frame-level, slice-level, macroblock-level and data-level into one implementation. Each level parallelism is designed in a hierarchical parallel framework and mapped onto the multi-cores and SIMD units on multi-core architecture. According to the analysis of coding performance on each level parallelism, we propose a method to combine different parallel levels to attain a good compromise between high speedup and low bit-rate. The experimental results show that for CIF format video, our method achieves the speedup of 33.57x-42.3x with 1.04x-1.08x bit-rate increasing on 8-core Intel Xeon processor with SIMD Technology.

Keywords: H.264 encoder; Hierarchical Multi-level Parallelisms; Multi-core Architecture.

1. Introduction

H.264 [1] as a video coding standard is now being used widely due to its high-quality video content and low bit-rate. However, it makes encoding process more complex and requires more computation than previous coding standards. Given fixed fidelity, H.264 reduces bit-rate up to about 50% at the cost of more than three times computational complexity compared to H.263 [2]. Therefore, the hardware and software co-design parallelisms are needed to accelerate the speed of encoder for real-time application. Multi-core processor architecture [3] is now becoming the mainstream solution for next generation general computation. Unlike the simultaneous multiple threading (SMT) [12] and hyper-threaded processor (HT) [13] where most micro-architectures are shared between logical processors, multi-core processor introduces new microprocessor technologies to deliver high computation ability. First, multi-core processor integrates multiple single processor cores into one chip which

supports the real coarse-grained hardware thread parallelism. Second, each core is equipped the SIMD instruction sets to provide the fine-grained parallelism. Third, each core has independent L1/L2 cache to increase the bandwidth and hit rate. All these features can be beneficial for improving the speed of H.264 encoder.

Many parallel algorithms for H.264 encoder were discussed in previous work. A parallel scheme is addressed in [4, 5] that encodes the slices of a frame in parallel on Intel hyper-threading architecture. It mainly concentrates on the slice parallelism based on fixed IBBP encoding structure. A method that utilizes the dependency of reconstructed macroblock (MB) and encoding macroblock to encode multiple macroblocks in parallel is reported in [6]. Another parallel algorithm for macroblock encoding is reported in [2, 11]. It uses approximate neighboring encoding information to find the optimal coding mode of the current coding block. A pipeline algorithm is discussed to parallelize macroblock analysis and the performance is analyzed on Cell processor in [7]. A H.264 decoder is implemented on general-purpose processors by using SIMD instructions in [8]. Parallel motion estimation scheme for H.264 are discussed in [9, 10].

We expand the method proposed in [4, 5] to multi-B frames in the frame level and combine frame, slice, macroblock and data parallelisms for H.264 encoder into one HMLP framework. First, the HMLP model and the design details of each level parallelism for H.264 encoder are presented. Then, based on performance analysis on each parallelism the tradeoffs between multiple parallel levels are attained to optimize the encoding performance.

The rest of this paper is organized as following. Section 2 provides detail design and implementation of our HMLP model for H.264 encoder. Section 3 demonstrates performance results and discusses the results. The selection strategy of multi-level parallelisms is illustrated in section 4. And section 5 concludes this paper.

2. Hierarchical Multi-level Parallel Parallelisms for H.264 Encoder

In H.264, a video sequence includes many frames. Each frame is partitioned into slices, which is the encoding unit and independent of other slices in the same frame. Slice can be decomposed into macroblock which is the unit of encoding algorithm. The structure above provides potential parallel optimization opportunities.

2.1. The Framework of HMLP model

As in Figure 1, the framework of HMLP model for H.264 encoder is designed to integrate four levels of parallelisms of frame, slice, MB and data into one implementation. It consists of encoding threads and queue buffers. Three

kinds of encoding thread – frame thread, slice thread and MB thread, do the encoding process at three different levels. Frame thread is on the top level. Frame threads create the threads for the slice-level which hierarchically create the threads for the MB-level. The data-level parallelism which acts as functional parallelism is included in the MB encoding thread. All above parallelisms compose a hierarchical parallelism tree, where from root node to the leaf node the parallelism grain is decreasing. The HMLP framework shows good scalability. In each parallel level, the size of the processing unit for each thread can be decreased to increase the number of thread. For example the frame can be decomposed into more slices to increase the slice encoding thread. For different levels, because of the hierarchical structure of frames, slice, MBs and data there are many parallel grains to select the size of processing unit.

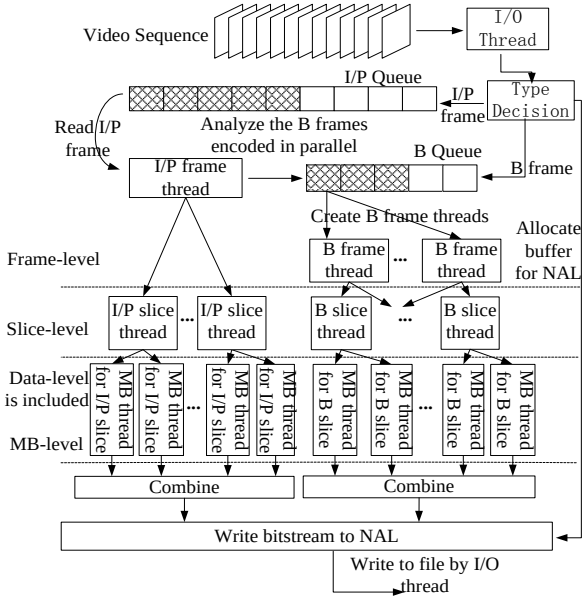


Fig. 1.. Hierarchical Multi-level Parallelisms (HMLP) framework for H.264 encoder.

2.2. Frame and Slice level Parallelisms Design and Implementation

Usually, a sequence of frames is encoded using an IB...BPB…BP… structure. The number of B frame between two P frames can be multiple. Here, I and P frames are treated as the reference frames and B frame are considered as non references in order to explore more parallelism. Figure 2 shows the principle of frame-level parallelism. The display order indicates the original order of video frame. The dependency between the frames is showed in the encoding order. In this encoding order, the completion of encoding a P frame will make the subsequent one P frame and some B frames ready for encoding

in parallel. Here, one P frame and the B frames in the same column will be encoded in parallel order.

H.264 encoder is divided into three parts: input processing, encoding and output processing. As depicted in figure 1, the input processing reads uncompressed images, decides type and allocates the NAL node for bit-stream. The output processing checks the NAL queue and writes the bit-stream after encoding to the output file. One I/O thread is used to handle the input and output processing. In order to explore parallelism in between frames, two queues are used, one is for I or P frames and another is for B frames. After each frame's type is decided the frame will be put into the corresponding queue. The I/P encoding thread will fetch an I or P frame from the I/P queue and check the B frames which are independent of current I/P frame and ready for encoding in the B queue. After that, I/P thread will create B frame encoding thread for each above B frame and encode the I/P frame with these B frames in parallel.

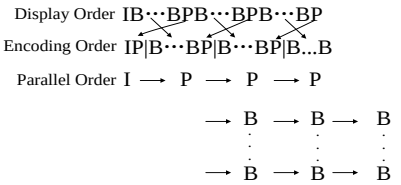


Fig. 2. The frame dependencies and parallel encoding order in H.264.

A frame can be divided into small slices which are independent and can be encoded in parallel. As figure 1 illustrates, each frame encoding thread like I/P thread and B thread divides the frame into slices and create encoding thread for each slice. After encoding, each slice thread writes the bit-stream to the NAL in the order of slice.

The pseudo code of frame-level and slice parallelisms is listed below. We use I/O thread to process the input and output and I/P thread to create B frame encoding threads dynamically to encode the frames in parallel. In each frame encoding, frame is partitioned into slices and slice encoding threads are created for each of it to encode. Finally, the bitstream is assembled and write to the file.

```
I/O thread Code:
while (input video sequence is not NULL) do
  if (there is free entry in image buffer) then
    read a frame to image buffer and decide its type;
    if (the type is I or P) then
      enter I/P queue;
    else
      enter B queue;
    end
    allocate a node in NAL queue for current frame;
  else if (there is bitstream node in the NAL queue)
    write the bitstream node to output file;
  else wait;
```

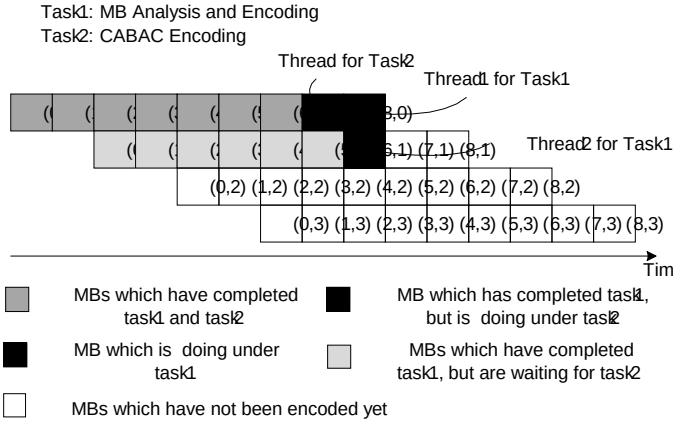
```
  end
end while
I/P thread Code:
while (true) do
  if (there is frame in the I/P queue) then
    fetch a frame from I/P queue;
    analyze the B frames in B queue;
    create Encoding thread for B frame
        which can be encoded in parallel;
    call Encoding thread to encode current frame;
  else if (all frames are encoded)  exit;
  else  wait;
  end
end while
Encoding thread Code:
for each slice in the frame
       create slice encoding thread to encode;
assemble the bitstream and write to file
```

2.3. MB and Data Level Parallelisms Design and Implementation

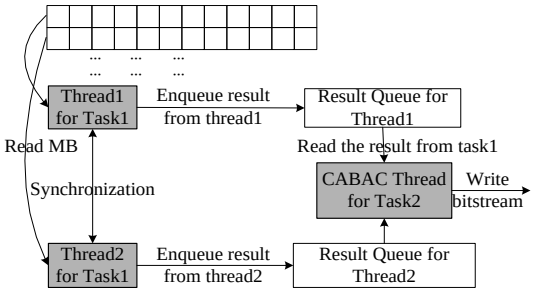
MB encoding process is the most time-cost part in H.264. In the implementation, it is composed of three modules – MB analysis, encoding and CABAC. MB analysis module mainly analyzes intra and inter prediction mode, predicts motion vector, and decides the MB type. And the MB encoding module mainly processes the DCT, quantization and de-blocking filtering. From the analysis in [6], the processing above indicates that the MB analysis and encoding of current MB depends on results of current MB's top and left neighboring MBs. The CABAC module depends on the CABAC result of last MB. So, it must be processed sequentially in the row order.

Figure 3(a) illustrates the principle of parallel MB encoding process in a slice structure that consists of 4x9 MBs. MB is indexed by the coordinate MB(i,j). According to the dependency, in order to analyze and encode MB(i,j), it is necessary to refer to the results of its left MB(i-1,j), top-left MB(i-1,j-1), and top MB(i,j-1). Therefore, the initial way is to analyze and encode two MBs, MB(0,0) and MB(1,0) sequentially. After this, MBs in each column such as MB(2,0) and MB(0,1) can be analyzed and encoded in parallel, meanwhile, MB(0,0) and MB(1,0) can do CABAC. And then MB(3,0) and MB(1,1) can be analyzed and encoded in parallel. In such a way, MB(6,0), MB(4,1), MB(2,2) and MB(0,3) can be analyzed and encoded in parallel. However, CABAC must be processed sequentially in the row order. According to the Amdahl's law, the total time is decided by the cost time of CABAC. The experiment shows the cost time of CABAC is about half of the sum of MB analysis and encoding time. So, as figure 3(a) illustrates, three threads are created for MB encoding process, two of which execute MB analysis and encoding (Task1) and one of which executes CABAC (Task2). The producing rate of two threads for Task1

is enough to match the consuming rate of one thread for Task2. In the parallelization pattern, each slice is partitioned into MB rows. Each thread for Task1 processes the interlacing MB rows in a slice – one thread processes odd rows the other processes even rows. Thread for Task2 processes MB rows in sequential and synchronize with the two threads executing Task1.



(a) Principle and task partition for parallel MB encoding process



(b) The implementation of MB-level parallelism

Fig. 3. The multi-threads implementation of macroblock-level parallelism

Figure 3(b) shows the implementation of MB-level parallelism. The slice thread created in section 2.2 is taken as the CABAC thread which creates two threads for MB analysis and encoding. Two queue buffers are used to store the results of MB analysis and encoding from the threads. CABAC thread reads the two queue buffers alternatively to encode and write the final results to bit-stream.

The SIMD technique can be used to speed up encoding process in the data-level. We use the SIMD instruction to rewrite the following encoding modules: integer DCT/IDCT transform, quantization, motion compensation, sub-pel search, de-blocking and SAD calculation. Because the SIMD is an instruction optimization technology, it does not compete with frame or slice parallelism for physical threads.

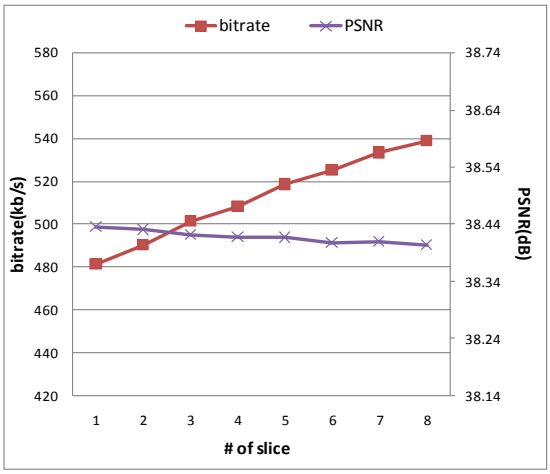
3. Experiments and Performance Evaluation

The experimental tests of multi-level parallel H.264 encoder is performed on 8 cores Intel Xeon processor running at 2.0GHz, 1M L2 Cache and supporting MMX/SSE1/SSE2. If it is unspecified, the test video is Foreman in CIF format (352x288) with 300 frames. The profile of H.264 encoder is main profile which is configured as following: (1) inter-coding using B-slices and weighted prediction; (2) deciding references on a per partition basis; (3) using hexagonal search; (4) using 1/4-pel resolution research (5) enabling all search types; (6) using CABAC.

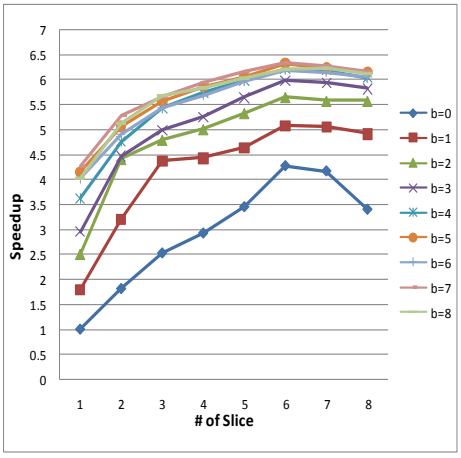
3.1. Coding Performance Versus Frame and Slice Parallelisms

The coding performance is one of the most important issues in the video coding. Even though parallelisms can make video data process faster, it must not significantly sacrifice the coding performance. Figure 4(a) shows the encoder performance when a frame is divided into different number of slices, here number of B-frame is 2. We can see that with 8 slices in each frame, we have a bit-rate increment close to 15% which is not admissible. Too much slice parallelism causes bit-rate rising. Thus, the slice parallelism is sensitive and restricted to bit-rate. Another quality parameter PSNR does not behave so adversely. It is seen that PSNR has a small variation around 38.42 dB. It is concluded that *bit-rate is the key coding performance parameter* that limits frame and slice parallelisms.

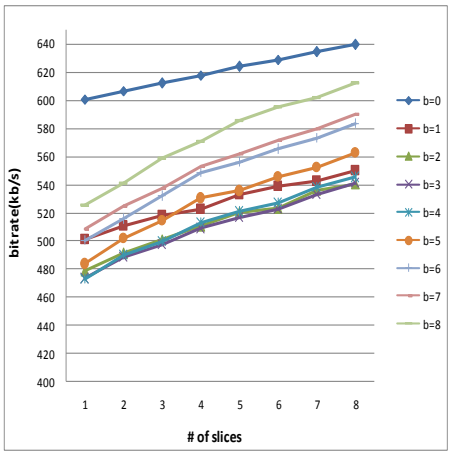
As mentioned early, B frame can be encoded with P frame in parallel, so multiple B frames can increase the degree of parallelism. But, it also influences the bit-rate and drops down the image quality because of the inaccurate bi-predictions. One challenge is to attain a high quality. So, the proper amount of B frame should be selected. Meanwhile, partitioning one frame into multiple slices can increase the degree of parallelism, but it also increases the bit-rate. Because it isolates the correlation between different slices in one frame and adds slice heads to the bit-stream. Thus, the amount of slice should be selected carefully as well. Figure 4 illustrates the speedup and bit-rate variation with of the number of B frames and slices. In figure 4 (b), There is a best speed up of 6x to 6.3x when the number of B frames ranges from 3 to 7 and the number of slices in each frame is 6. In figure 4 (c), the bit-rate descends about 110kb/s when the number of B frames ranges from 0 to 3 and rises up about 50 kb/s when B frames varies from 3 to 8. Thus, considering frame level only, best speedup and relative lower bit-rate are achieved when the number of B frames ranges from 2 to 3. On the other side, given 2 or 3 B frames, the bit-rate increases almost linearly with the number of slices. The bit-rate increases about 40kb/s compared with no slices partition when the number of slices reaches 6, at which the best speedup is attained. *The important observation is that setting the number of B frames to 2 to 3 and partitioning a frame to 6 slices delivers the best tradeoff for frame and slice parallelisms that achieves a 6.0x-6.3x speedup with 1.08x bit-rate.*



(a) bit-rate and PSNR vs. the number of slices in a frame



(b) speedup



(c) bit-rate

Fig. 4. Speedup and coding performance of frame–level and slice-level parallelisms.

3.2. Coding Performance Versus MB and Data Parallelisms

As mentioned before, MB-level parallelism utilizes the inherent dependencies of different MB encoding processes in a slice. Thus, it can increase the degree of parallelism while keeping the bit-rate no changing. Figure 5 shows the speedup of adding the MB-level parallelism to frame-level and slice-level parallelisms. Comparing to the number of 6 slices where the best speedup is achieved in figure 4 (b), the best speedup in figure 5 shifts to point of 3 slices. Meanwhile, the speedup of frame-level parallelism almost keeps the same.

MB-level parallelism doesn't increase the bit-rate. As refers to figure 4 (c), the bit-rate decreases about 20 kb/s when the number of slices reduces from 6 to 3 where the peak speedup is achieved. *One important conclusion is that MB-level parallelism decreases the bit-rate while keeps the best speedup of 6.x through reducing the number of partitioned slices and increasing the MB parallelism in a frame.* It is observed that when number of B frame is 2 to 3, the partitioned slices in a frame is 3 and the MB-level parallelism is used we can achieve a good speedup and maintain a lower bit-rate.

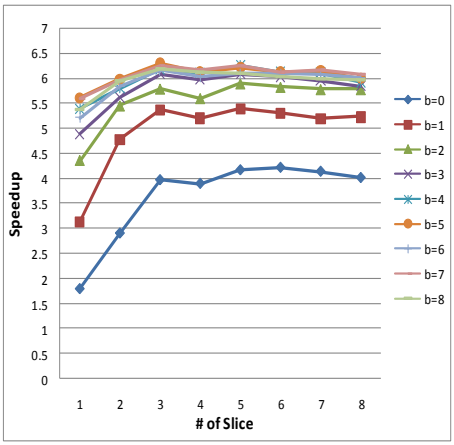


Fig. 1. Speedup of adding MB-level parallelism to frame-level and slice-level

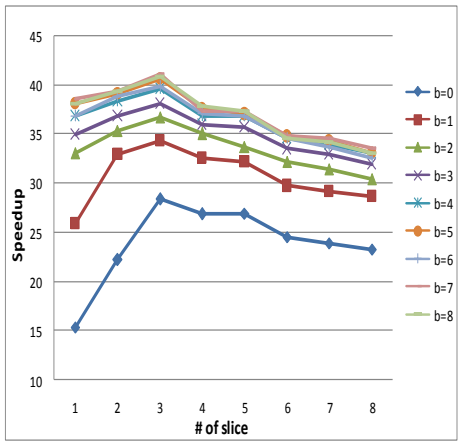


Fig. 2. Speedup of combining four level parallelisms of frame, slice, MB and data.

Data-level parallelism utilizes the SIMD instruction to improve the computation of encoding process especially the vector and matrix computation. Thus, it will not increase the bit-rate as well. As figure 6 illustrates, about 42.3x speedup is achieved with 1.04x bit-rate arising by combining four level parallelisms.

3.3. Performance Comparison with Other Related Works

Table 1 shows the performance comparison between our hierarchical multi-level parallelism H.264 encoder with other related works. In [5], slice-level parallelism is used to a fixed frame structure. For 2 B frames the speedup of 4.31x-4.69x is achieved on 4 Intel Xeon processors with Hyper-Threading Technology (8 logical processors). This method is implemented in our test bed and achieves the speedup of 5.56x-5.72x while with 1.11x bit-rate (ratio). Single marcroblock-level parallelism method in [6] and single data-level parallelism method in [8] achieve the speedup of 3.08x and 2x-4x separately, and keep the bit-rate no change. Comparing with above method, for 2 B frames structure, our hierarchical multi-level parallelisms method gains the

speedup of 33.57x-34.78x while with 1.05x bit-rate (ratio). Our multi-level method replaces part slice-level parallelism with the macroblock-level parallelism to reduce the number of slices.

Table 1. Performance comparison with other works

Parallelization method	Speedup	Bit-rate(ratio)
Slice-level with fixed B frame[10,18]	5.66x-5.72x	1.11x
Single MB-level parallelism[5]	2x-4x	1x
Single data-level parallelism[17]	3.08x	1x
Hierarchical multi-level parallelisms in our work	33.57x-34.78x	1.05x

4. Conclusions

H.264 provides many potential parallel optimization opportunities. Single level parallelism scheme can speed encoding, however, it achieves low speedup and increases the bit-rate. A hierarchical multi-level parallelisms design for H.264 encoder is presented which exploits the multi-level parallelisms of frame, slice, macroblock and data in one implementation on multi-core architecture. The tradeoffs of integrating multiple levels are analyzed to gain good speedup and also to keep bit-rate and the video degradation as minimal as possible. The speedup of 42.3x is achieved on 8 Intel SIMD processors with SIMD Technology The method demonstrated can also be applied to other video coding and parallel hardware.

5. Acknowledgement

This paper is financially supported by the National Natural Science Foundation of China under Grant No.60703049; Intel grant for a study of multi-core programming environment.

6. References

1. T. Wiegand, G. J. Sullivan, G. Bjontegaard et al.: Overview of the H.264/AVC video coding standard. IEEE Transactions on Circuits and Systems for Video Technology, Vol. 13, No. 7, 560-576. (2003)
2. T-C. Chen, Y-W. Huang, and L-G. Chen.: Analysis and design of macroblock pipelining for H.264/AVC VLSI architecture. In Proceedings of 2004 International Symposium on Circuits and Systems, Vol. 2, 273-276. (2004)

3. Jeff Parkhurst, John Darringer, PBill Grundmann.: From Single Core to Multi-Core: Preparing for a new exponential. In Proceedings of the 2006 IEEE/ACM International Conference on Computer-aided Design, 67-72. (2006)
4. Chen Yen-Kuang, Tian Xinmin, Ge Steven and Girkar Milind.: Towards Efficient multi-level threading of H.264 encoder on Intel Hyper-Threading Architectures. In Proceeding of International Parallel and Distributed Processing Symposium. Vol.18: 889-898, (2004)
5. Ge S., Tian X., and Chen Y.-K.: Efficient Multithreading Implementation of H.264 Encoder on Intel Hyper-Threading Architectures. In Proceeding of IEEE Pacific-Rim Conference on Multimedia, 469-473 (2003)
6. Zhao Zhuo and Liang Ping.: A highly efficient parallel algorithm for H.264 video encoder. In Processing of 2006 IEEE International Conference on Acoustics, Speech, and Signal Processing, Vol. 5, 489-492 (2006)
7. Jonghan Park and Soonhoi Ha.: Performance Analysis of Parallel Execution of H.264 Encoder on the Cell Processor. In Proceedings of the 2007 IEEE/ACM/IFIP Workshop on Embedded Systems for Real-Time Multimedia, 27-32, (2007)
8. Zhou X., Li E. Q., and Chen Y.-K.: Implementation of H.264 Decoder on General-purpose Processors with Media Instructions. In Proceeding of SPIE Conference on Image and Video Communications and Processing, 224-235, (2003)
9. Chuan-Yiu Lee, Yu-Cheng Lin, et al.: Multi-pass and frame parallel algorithms of motion estimation in H.264/AVC for generic GPU. In Proceeding of International Conference on Multimedia & Expo, 1603-1606, (2007)
10. Lin Chia-Chun, Lin Yu-Kun and Chang Tian-Sheuan.: PMRME: A Parallel Multi-Resolution Motion Estimation algorithm and architecture for HDTV sized H.264 video coding. In Proceeding of IEEE International Conference on Acoustics, Speech and Signal Processing, Vol. 2, 385-388, (2007)
11. Y.-W. Huang, T.-C. Chen, C.-H. Tsai, et al.: A 1.3TOPS H.264/AVC single-chip encoder for HDTV applications. In Proceedings of International Conference on Solid-State Circuits, 128-130. (2005)
12. D. M. Tullsen, S. J. Eggers, H. M. Levy.: Simultaneous Multithreading: Maximizing On-Chip Parallelism. In Proceeding of International Symposium on Computer Architecture, 392-403, (1995)
13. D. Marr, F. Binns, D. L. Hill, G. Hinton, D. A. Koufaty, J. A. Miller, and M. Upton.: Hyper-Threading Technology Micro-architecture and Architecture. http://developer.intel.com/technology/itj/2002/volume06issue01/art01_hyper/vol6iss1_art01.pdf (2002)

Haitao Wei is currently a Ph.D candidate at Huazhong University of Science and Technology in China. His research interests include parallel computing, compilation optimization, and parallel programming model.

Junqing Yu is the corresponding author of this paper. He received his PhD in Computer Science from Wuhan University in 2002. He is currently an Associate Professor at the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China. His research interests include digital media processing and retrieval, multi-core programming environment.

Jiang Li: was born in 1984. He got his master degree from Huazhong University of Science and Technology in 2008. His research interests include digital media processing and parallel computing.

Received: May 02, 2009; Accepted: August 13, 2009.