

Parallelization of H.264 Deblocking Filter on IBM Cell Broadband Engine

劉君翊

Grad. Inst. of Networking and Multimedia
National Taiwan University
Taipei, Taiwan

呂敏中

Dept. of CSIE
National Taiwan University
Taipei, Taiwan

ABSTRACT

With the ever-growing size of display devices and demands for higher video resolution and quality from consumers, newer video codecs are continued to be developed for higher compression rate. The latest one is the H.264 video compression standard, having grown popular since its release and been adopted for the Blu-Ray and YouTube videos for delivery of Full High-Definition contents. However, with the higher compression rates come the increasing complexity of the codecs and needs for higher computation power. As video encoding/decoding processes are generally friendly to parallelization, they are usually parallelized to run on multi-core platforms to gain higher computation throughput. In this work, we modify FFMpeg multi-media framework and parallelize *deblocking filter*, a time-consuming yet easily parallelizable part of the H.264 decoding process, on IBM Cell Broadband Engine. Evaluation shows our parallelization technique brings slight improvement on total decoding time.

1 INTRODUCTION

The demands for higher video quality have encouraged more and more complex encoding/decoding techniques, such as the H.264 video compression standard. However, the more complexity of the codecs also means more needs for more computation power. As single-core technology advances like instruction-level parallelism, superscalar and clock rate scaling have come to their end, programmers have to seek for other means in order to achieve high computation throughput. One viable and promising candidate is heterogeneous multi-core platforms, which offer good performance-per-watt values. In this work, we parallelize a part of H.264 decoding process on IBM Cell Broadband Engine, a popular heterogeneous multi-core architecture, to speed up a H.264 decoder readily available on the internet.

1.1 H.264 Video Compression Standard

The H.264 video compression standard is the latest video compression standard to date, possessing better compression rate and higher compressed video quality than

previous video compression standards, and is currently utilized in Blu-Ray videos and YouTube, to name a few. However, with the higher compression rate comes with the more decoding complexity than previous codecs. There are five major procedures in an H.264 decoding process:

Entropy Coding: Carries out lossless decompression of a video; includes variable length coding and arithmetic coding.

Intra Prediction: Predicts the currently-decoding macroblock's values from nearby macroblocks. The decoding process goes from top to bottom and left to right, so the four macroblocks from left, top, top-left and top-right are referenced.

Inter Prediction: References the information from previously-decoded frames.

Inverse Transform: Inverses the process which is used to increase compression rate by mathematical transforms on macroblock data.

Deblocking Filter: Rids the decoded frame of sharp edges among macroblocks to improve the visual quality of the video.

Current proposed techniques to parallelize H.264 encoding/decoding process generally boil down to three categories: GOP-level, slice-level and macroblock-level.

GOP-level: In an H.264 video, every GOP (group of picture) is independent from each other and may be processed in parallel. This is the easiest approach of parallelization, but when the video resolution is large, a huge amount of memory accesses are required.

Slice-level: A frame in an H.264 video can be partitioned into several slices, which are independent from each other and may be processed in parallel. However, the number of slices in each frame can be little, resulting in low degree of parallelization. Yet, the more slices a frame has, the lower the compression rate is. Also, the workload of different slices can differ greatly.

Macroblock-level: A frame is composed of many macroblocks, which may be processed in parallel as long as their dependencies are resolved, as in *intra prediction* and *deblocking filter*, as explained in Section 2. This level gives better degree of parallelization but requires more efforts for such.

[†] A video frame is partitioned into macroblocks, each of 16 pixels in width and 16 pixels in height. Many kinds of encoding/decoding operations operate on a macroblock as the basic unit.

Readers are referred to [1] and [2] for more complete introduction to the H.264 video compression standard.

1.2 IBM Cell

The Cell Broadband Engine Architecture (often abbreviated as Cell) jointly designed by Sony Computer Entertainment, Toshiba and IBM, is a heterogeneous multi-core architecture with 64-bit PowerPC instruction set, consisting of one Power Process Element (PPE) and eight Synergistic Processing Elements (SPEs) all running at 3.2GHz clock rate and provides computation throughput over 100 GFLOPS. The PPE is generally considered suitable for a variety of computational workloads, and SPEs, with its RISC-like architecture and 128-bit SIMD capability, are generally considered suitable for arithmetic workloads. We refer our readers to [3], [4], and [5] for detailed information of this architecture.

1.3 Motivation

We profiled *FFMpeg* [6], a popular and open-source multi-media framework, which is already equipped with a H.264 decoder, for insights on what parts of the H.264 decoding process are most time-consuming. The profiling results are in Figure 1. As illustrated, the *deblocking filter* process consumes around 30% time of H.264 decoding, and we would like to parallelize this part of H.264 decoding process to improve its performance, eyeing the its macroblock-level parallelism.

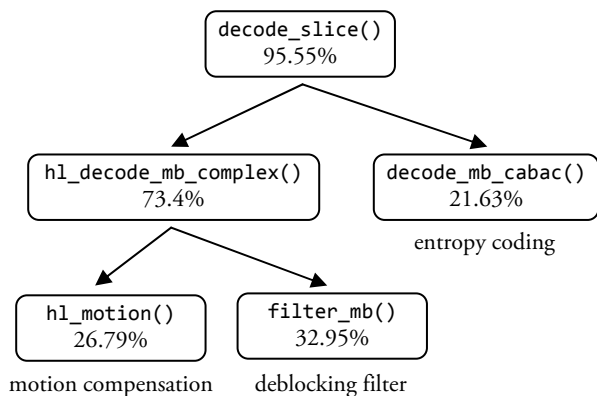


Figure 1. The portions of different procedures' time consumption of FFMpeg's H.264 decoder on x86

This report is organized as follows. In the next section we introduce our parallelization methodology. We evaluate and analyze the performance of our methodology in Section 3. Section 4 briefs on related works on the parallelization of H.264 encoding/decoding process. Our work is concluded in Section 5 where we also share our ideas for future work and other thoughts.

2 METHODOLOGY

FFMpeg implements the H.264 decoding process somewhat differently from the H.264 specification. For example, the standard specifies that *deblocking filter* should be applied to a frame after the whole of it has been decoded. However, FFMpeg does *deblocking filter* of a macroblock as soon as it has been decoded. It stores a non-deblocked version of the macroblock separately for *intra prediction's* reference. It is worth noting that this modification of FFMpeg enables us to parallelize *deblocking filter* in a finer granularity than frame-based, to the granularity of macroblock-based.

Here we propose two approaches to parallelize deblocking filter. The first approach is finer-grained, to the granularity of macroblocks; however it failed to work with deadlocks and race conditions we were unable to resolve, and is included here only for completeness. The second approach proposed here is coarser-grained, to the granularity of rows of macroblocks. Both the approaches let the PPE to carry out all the decoding process except deblocking filter, for which SPEs are responsible.

2.1 The First Approach

The first approach is supposed to work as followed. After PPE has processed a row of macroblocks (with all the decoding workload other than deblocking filter), it signals an SPE, through mailboxes, to process the row of macroblocks. The SPE, on receiving the signal, will fetch the row through DMA. Since in deblocking filter, a macroblock depends on one to the left of it and one to the top of it, An SPE can start processing a macroblock in the row only after the depended macroblock has been processed by another SPE. As such, an SPE also signals PPE about its decoding progress (that is, to which macroblock in the row it has processed) so that PPE can tell the dependent SPE about to which macroblock the dependent SPE can fetch from the depended SPE and process. After the row of macroblocks is processed by an SPE, it is DMA'ed back to the main memory. As SPEs are processing the rows, PPE can continue to process other rows. The communication graph of PPE and SPEs is depicted in Figure 2. In essence, the deblocking process would progress from the top-left corner of a frame diagonally to the bottom-right corner, due to the nature of inter-macroblock dependence.

2.2 The Second Approach

As the first approach did not work, we resorted to the second approach which uses a coarser granularity. This approach works as followed. After PPE has processed a row of macroblocks (again, with all the decoding workload other than deblocking filter), it signals three SPEs, again through mailboxes, to process the row of macroblocks. As the three color components (Y, Cb, Cr)

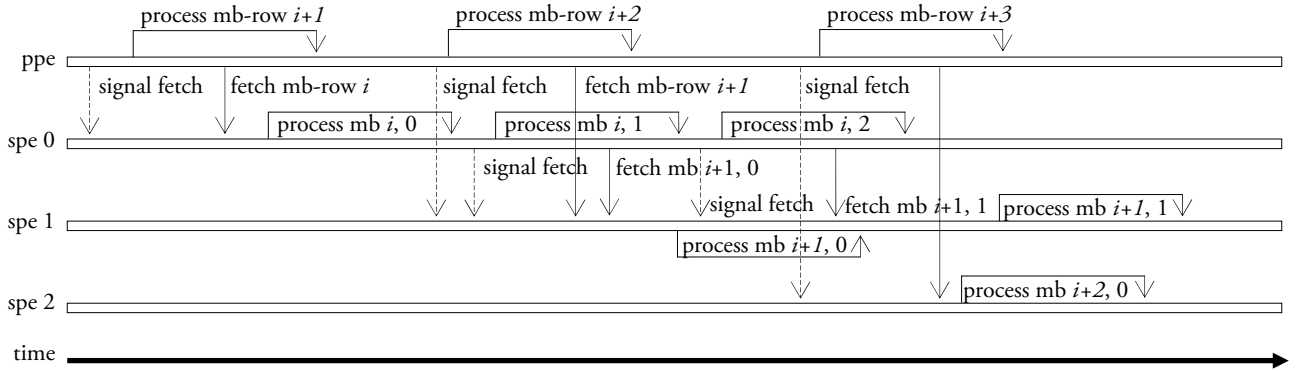


Figure 2. The communication between PPE and SPEs in the first approach. Suppose PPE has just processed mb-row 0 (the 0th row of macroblocks); mb x, y denotes the y^{th} macroblock in the x^{th} row of macroblocks.

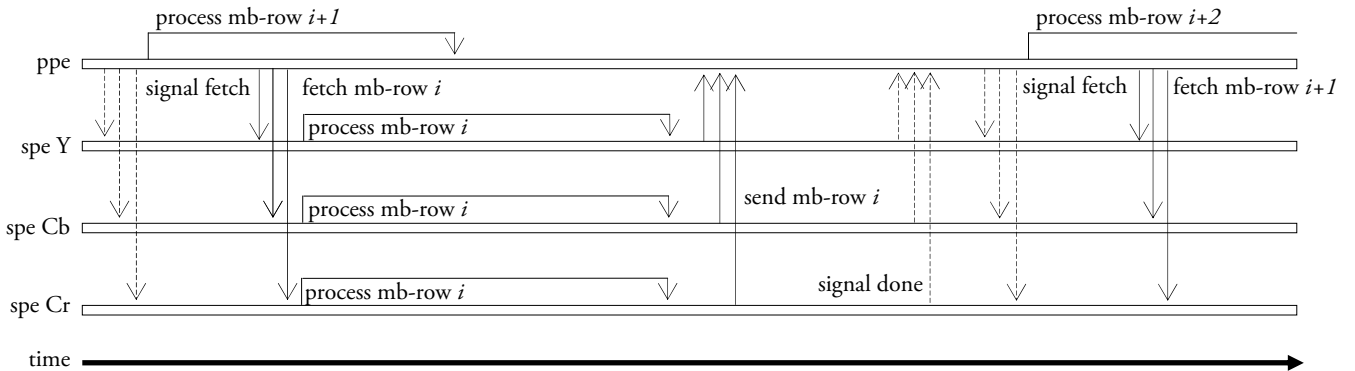


Figure 3. The communication between PPE and SPEs in the second approach. Suppose PPE has just processed mb-row 0 (the 0th row of macroblocks).

of a frame can be processed separately, each of the three SPEs can process a color component of the row of macroblocks independently[†]. Again, as SPEs are processing the row, PPEs can continue to process another row of macroblocks. After SPEs finish processing a row of macroblocks, it is DMA’ed back to the main memory. The communication graph of PPE and SPEs is depicted in Figure 3.

3 EVALUATION AND ANALYSIS

We evaluated the FFMpeg H.264 decoder modified according to our second approach with SPE-supported deblocking filter processing, on an IBM Cell blade consisting of a Cell Broadband Engine running at 3.2GHz, with 2GB main memory. We used two video clips available from H264info.com [7] for evaluation, which are listed in Table 1.

Table 1. The test video clips

Video name	Fantastic Four: Rise of the Silver Surfer (trailer)	Serenity (trailer)
Resolution	1280× 544	1280×720
Frame count	3096	3347
Length	129.13 sec	139.60 sec

The total runtime (“real”), user-mode time (“user”) and kernel-mode time (“sys”) to decode the videos with

the original and the parallelized FFMpeg are listed in Table 2. We can see that while the total runtime has not decreased apparently, the user-mode time has decreased by about 5%, while kernel-mode time has significantly increased by an order of magnitude.

Table 2. Runtime of the original and parallelized decoder

Video	Time	real	user	sys
Fantastic Four	original	4m57s	4m55s	2 sec
	parallelized	4m54s (-1%)	4m39s (-5.4%)	15 sec
Serenity	original	5m56s	5m54s	2 sec
	parallelized	5m56s (±0%)	5m35s (-5.4%)	21 sec

We suspected the increase of kernel-mode time was due to mailbox communication, and then ran the IBM Full-System Simulator for Cell [8] in hopes for further insight. We ran the simulator with our modified FFMpeg decoder in cycle-accurate mode (for PPE) and pipeline mode (for SPE) with a smaller and shorter (192×82 in dimension and 14 seconds in length) video clip, which should normally take just a second or two to decode. We found out that SPEs have spent a large portion of its runtime stalling to wait for DMA to be done and for mailbox messages. The SPE responsible for the Y

[†] Readers might wonder that the Y component, with its macroblock 4 times as detailed as Cb and Cr components as defined in the H.264 4:1:1 YCbCr color format, would require much more processing than the Cb and Cr components, resulting in work imbalance. However, this is not entirely true: as Cb and Cr macroblocks are visually larger, they would require more deblocking filter processing.

component ran for a total of 4.209×10^9 cycles, yet, 4.173×10^9 cycles of them were “channel stall cycles” (channel stall is due to blocking by reading DMA status, reading an empty mailbox or writing to a full mailbox). Only 1.759×10^7 cycles were actually for computation, and another 1.757×10^7 cycles for dependency stalls and branch miss stalls.

We then added timing functions to the main program running on PPE to get to know how much time the PPE spent on mailboxes. As a result, the large kernel-mode time shown in Table 2 was revealed to have 99.8% of it being spent on PPE’s mailbox-related operations. The initialization and destroying of SPEs only contribute to about 0.5 milliseconds. We thus came to think that mailboxes posed a major hindrance for us to achieve enough performance gain.

4 RELATED WORKS

[9] also implements macroblock-level parallelization of H.264 decoder, letting the PPE only do the *entropy coding* process and off-load all other decoding processes to SPEs. It has devoted a large portion of text to how to selectively and dynamically allocate an SPE’s local storage space for codes, stack and heap as the storage has insufficient space (only 256KBytes) for the entire execution context. [10] uses slice-level parallelization for H.264 decoding, on a different platform. It also implements energy-efficient scheduling algorithms on the decoder for energy efficiency.

5 CONCLUSION, IDEAS FOR FUTURE WORK AND THOUGHTS

In this work, we parallelize the deblocking filter part of FFMpeg’s H.264 decoder on IBM Cell platform by off-loading the deblocking filter to SPEs with each responsible for a color component. PPE and SPEs can operate on different data simultaneously. The parallelization sees slight improvement on total decoder runtime and improvement of 5% in user-mode time, with the increase of kernel-mode time of an order of magnitude.

As explained in Section 3, the current reason for the little performance gain is mailboxes. If possible, we would like to revise our communication mechanism to drastically reduce mailbox usage. Actually, the enormous time spent on mailboxes could also indicate that our failed first approach would probably not deliver good performance boost despite its finer parallelization granularity, as it would require much more mailbox messages due to the fact that SPEs would have to communicate with each other through PPE and mailboxes about the progress of processed macroblocks.

We have encountered some issues related to profiling. The profiling results in Figure 1 are obtained on an x86 platform instead of a Cell platform. We actually did have

two profilers on our IBM Cell blade, *Valgrind* [11] and *gprof* [12]. However, the two profilers gave us inconsistent results: Valgrind indicated that some functions ran longer than `main()`, and gprof had a very low sample rate incapable of catching function calls accurately. In the end we had to profile FFMpeg with Valgrind on an x86 platform hoping that the results would not differ a lot from those on a Cell platform.

REFERENCES

- [1] T. Wiegand, G. J. Sullivan, G. Bjøntegaard, and A. Luthra. In *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, Jul. 2003
- [2] G. J. Sullivan and T. Wiegand. Video Compression – From Concepts to the H.264/AVC Standard. In *Proceedings of the IEEE*, vol. 93, no. 1, Jan. 2005
- [3] J. A. Kahle, M. N. Day, H. P. Hofstee, C. R. Johns, T. R. Maeurer, and D. Shippy. Introduction to the Cell multiprocessor. In *IBM Journal of Research and Development*, vol. 49, no. 4/5, Jul. 2005.
- [4] H. P. Hofstee. Power Efficient Processor Architecture and the Cell Processor. In *Proceedings of the 11th International Symposium on High-Performance Computer Architecture*, Feb. 2005
- [5] H. Gschwind, H. P. Hofstee, B. Flachs, M. Hopkins, Y. Watanabe, and T. Yamazaki. Synergistic Processing in Cell’s Multicore Architecture. In *IEEE Micro*, vol. 26, no. 2, Mar. 2006
- [6] <http://ffmpeg.org/>
- [7] <http://www.h264info.com/>
- [8] <http://www.alphaworks.ibm.com/tech/cellsystemsml/>
- [9] M. A. Baker, P. Dalale, K. S. Chatha, and S. B. K. Vrudhula. A scalable parallel H.264 decoder on the cell broadband engine architecture. In *Proceedings of the 7th IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis*, Oct. 2009
- [10] Y.-H. Wei, C.-Y. Yang, T.-W. Kuo, S.-H. Hung, and Y.-H. Chu. Energy-Efficient Real-Time Scheduling of Multimedia Tasks on Multi-Core Processors. To appear in *the 25th Symposium on Applied Computing*, Mar. 2010
- [11] <http://valgrind.org/>
- [12] <http://www.gnu.org/software/binutils/>