International Journal of Multimedia and Ubiquitous Engineering
Vol.9, No.4 (2014), pp.221-226
http://dx.doi.org/10.14257/ijmue.2014.9.4.23

Implementation of GP-GPU with SIMT Architecture in the Embedded Environment

Kwang-yeob Lee and Jae-chang Kwak[1]*

Dept. of Computer Engineering, Computer Science, Seokyeong Univiersity, Jeongneung 4-dong, Seongbuk-gu, Seoul, Korea*

*kylee@skuniv.ac.kr, jckwak@skuniv.ac.kr**

Abstract

Recent embedded processors become to be multi-cored, due to the increased power consumption by higher operating frequencies. Multi-core processors stimulate applications to be parallelized. Since general purpose CPU has small number of core, which is optimized for serial processing, it has a limitation of parallel processing. To overcome this limitation, GPU is used for the parallel processing. In this paper, we implement GP-GPU of SIMT architecture for parallel processing in the embedded environment. The performance of the implemented GP-GPU is compared with the existing multi-core CPU of the embedded environment. The comparison results show the performance of parallel processing with the implemented GP-GPU is improved significantly.

Keywords: *GP-GPU, SIMT, parallel processing, embedded environment*

1. Introduction

The performance of processors has been improved by increasing operating frequencies based on speeding up transistors. In the 90nm manufacturing process, since the increased power consumption limits raising the operating frequency, processor designers are started to increase the number of cores to improve the performance. Multi-core processors stimulate applications to be parallelized. Since general purpose CPU has small number of core, which is optimized for serial processing, it has a limitation of parallel processing. To solve this limitation, GPU is used for the parallel processing.

GPU (Graphic Processing Unit) is kept developed up to GP-GPU (General Purpose Graphic Processing Unit). GP-GPU is getting to include most of techniques of existing CPU. GPU is not only for graphic data processing, but also for general purposes. GPU is composed of simple structured efficient cores. The number of cores is from tens up to thousands. These cores are suitable for the parallel processing [1].

Recently, applications with parallel processing are appeared in the embedded environment. Since the CPU used for the embedded environment has low operating frequency and little number of cores, it has a limitation of parallel processing. In this paper, we design and implement GP-GPU of SIMT (Single Instruction Multiple Threads) [2, 3] architecture for parallel processing in the embedded environment. In order to verify the performance, the image processing application is parallelized. The performance of the implemented GP-GPU is compared with the existing multi-core CPU of the embedded environment. The comparison

[1] Corresponding author

ISSN: 1975-0080 IJMUE

results show the performance of parallel processing with the implemented GP-GPU is improved significantly.

2. SIMT GP-GPU architecture

The structure of the implemented GP-GPU is shown in Figure 1. Each core has 16 stream processors (SPs). Each SP has three ALUs, so that it can process three instructions at the same time. Each thread (SP) is grouped on the basis of Warp [2]. Each Warp has maximum 16 threads. Since the implemented GP-GPU is a superscalar structure, maximum two instructions can be patched per Warp. For a single SP, an odd warp and an even warp are assigned and processed.

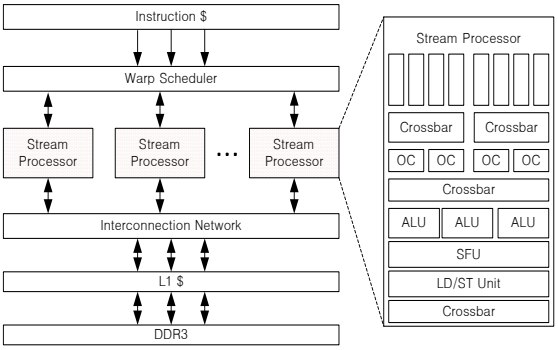

Figure 1. GPU High level architecture

The GPUs with the Single Instruction Multiple Data (SIMD) structure improved its performance by parallelizing SPs. [4-7] Each SP with the SIMD structure patches a single instruction and processes several data simultaneously. However, a module that can control the execution flow, by enabling every SP to independently execute different instructions, should exist as much as the number of SPs. Meanwhile, for the GPU with the SIMT structure, every SP processes the same one instruction. As shown in Figure 2, a single SP control unit can control every SP, reducing the use rate of hardware resources and power consumption.

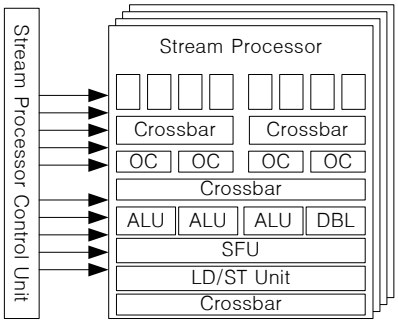

Figure 2. SP Control method

2.1. Superscalar instruction issue

As shown in Figure 3, the Warp Scheduler in the implemented GP-GPU patches four instructions by choosing an odd warp and an even warp among current enabled Warps. The implemented GP-GPU is designed to have three ALUs and one LD/ST unit, considering the usage frequency of instructions. Thus the superscalar mode must be arbitrated so that patched

instructions are not exceed four ALU instructions or two LS/ST instructions in the Warp scheduler. When the enabled Warp has the PC value, which references both instructions as LS/DT instruction, or when two enabled Warp try to patch only ALU instruction, it should be arbitrated to patch only three instructions, not four instructions.

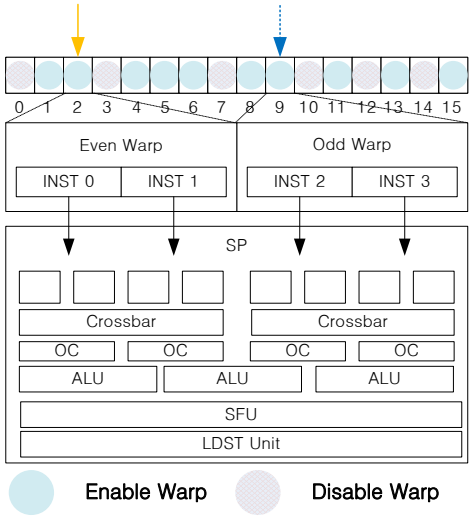

Figure 3. Warp arbitration and superscalar issue

2.2. Register file access

Since the processor with the superscalar structure patches more than two instructions at the same time, Register bank conflict can be occurred, as shown in Figure 4(A). When the register, which is referenced by each four operands, is located at the same bank, bank conflict is occurred and causes the pipeline stall. In order to solve this conflict, four read ports of register bank are required. But Increasing read ports is not an appropriate solution in the embedded environment, because of larger hardware scale. Another solution is to design the 6 stage operand collector [8], as shown in Figure 5(A). The 6 stage operand collector may waste the clock cycle in case of Figure 4(B), when each operand refers different register bank each other. Also, Flip-Flip waste can be occurred by the increased pipeline stage.

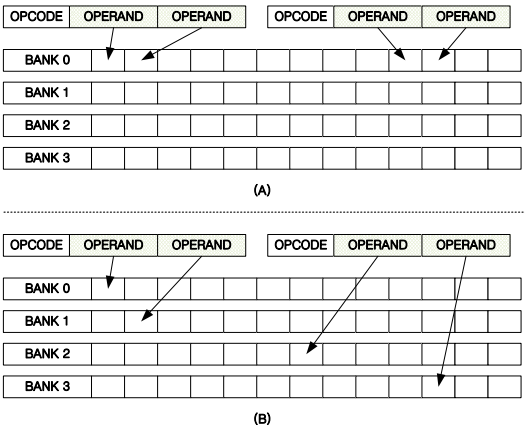

Figure 4. (A) Bank conflict (B) None bank conflict

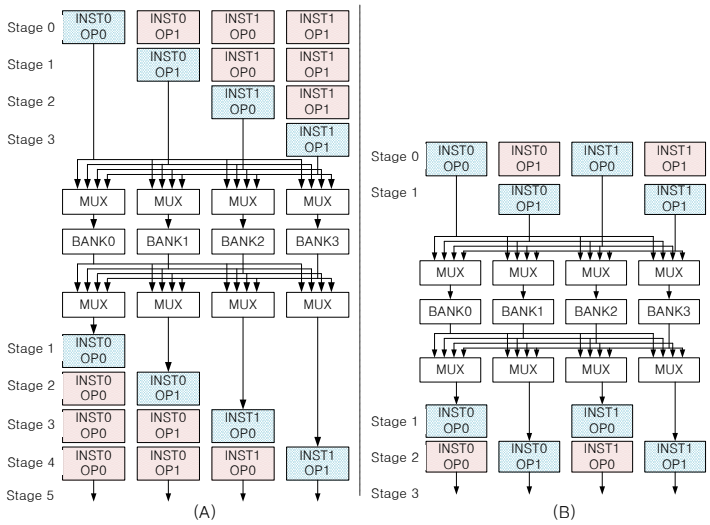

Figure 5. (A) 6 Stage operand collector, (B) 4 Stage operand collector

In this paper, the 4 stage operand collector is proposed, as shown in Figure 5(B). Both 6 stage and 4 stage operand collectors are implemented, and compared based on resource usage and consumption cycle. The results are shown in Table 1. In the comparison, the referenced register number is randomly generated for the operand of 8192 * 2 instructions. 4 stage operand collector can process all instructions with less Flip-Flop bits and fewer clock cycles.

Table 1. Comparison of operand collectors

	Used Flip-Flop Bits	Consumption Cycle
6 Stage	7728	14930
4 Stage	5156	13141

3. Experimental results

As the experimental environment, the VC707 FPGA Platform by Xilinx was used. Implemented GP-GPU has one core with 16 Stream Processors. The operating frequency in the platform is 50 MHz. A verification application was used for parallelizing an integral image creation algorithm, which is frequently used in the field of image processing and recognition, and the Gaussian Filter Mask algorithm. For parallelization of the CPU in the embedded platform, OpenMP was used.

Table 2. Processing time of integral image creation (unit: ms)

	1 Core	2 Core	3 Core	4 Core	Frequency
ARM Cortex-A15	19.46	11.43	8.19	7.72	1.6 GHz
ARM Cortex-A9	24.77	14.83	12.63	9.93	1.7 GHz
ARM Cortex-A9	31.24	19.46	14.57	11.46	1.4 GHz
ARM1176JZF	318.14	N/A	N/A	N/A	700 MHz
Proposed GPU	192	N/A	N/A	N/A	50 MHz

The size of image, used in the experimentation, is 640 x 480. It has 307,200 pixels. The image of 307,200 pixels was converted into the integral image. The experiment result is

presented in Table 2. As runtime difference occurred because of the different core operating frequency of each platform used in the experiment, clocks used for processing a single pixel were compared; the experiment result is shown in Figure 7(A) (The clock used for processing a single pixel = operating frequency × runtime/number of pixels) [10].

Table 3 shows the comparison of runtime with 3x3 Gaussian filter mask. Figure 7(B) shows the comparison of clock cycles used for processing a single pixel with 3x3 Gaussian filter mask.

Table 3. Processing time of 3x3 gaussian filter mask (unit: ms)

	1 Core	2 Core	3 Core	4 Core	Frequency
ARM Cortex-A15	27.61	13.93	9.57	7.81	1.6 GHz
ARM Cortex-A9	45.79	26.13	17.59	13.82	1.7 GHz
ARM Cortex-A9	50.37	37.59	30.59	25.72	1.4 GHz
ARM1176JZF	173.57	N/A	N/A	N/A	700 MHz
Proposed GPU	231.76	N/A	N/A	N/A	50 MHz

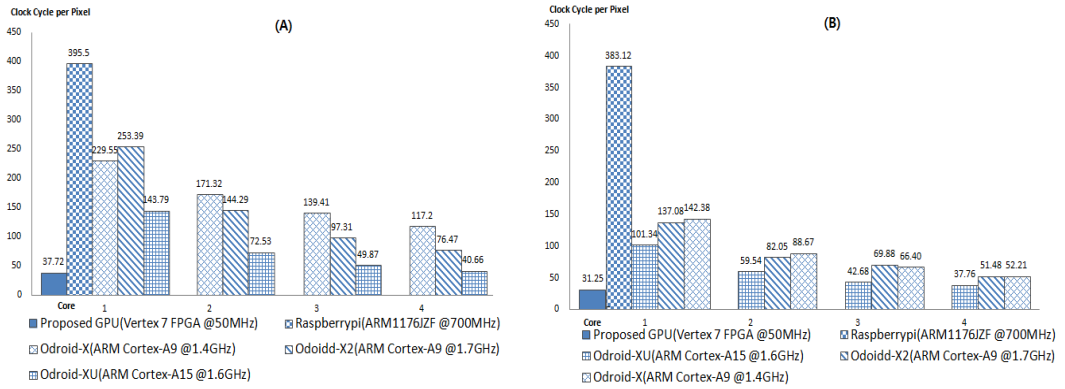

Figure 6. (A) Comparison of clock cycles used for processing a single pixel in case of integral image creation algorithm, (B) Comparison of clock cycles used for processing a single pixel in case of 3x3 gaussian filter mask

4. Summary

The multi-cored GP-GPU of SIMT architecture is designed and implemented for parallel processing in the embedded environment. Each core has 16 stream processors (SPs). The implemented GP-GPU can process maximum 4 instructions at the same time. 4 stage Operand Collector is proposed to reduce bank conflicts. The performance of the implemented GP-GPU is compared with the existing multi-core CPU of the embedded environment. The image of 640 x 480 sized 307,200 pixels was converted using three image processing algorithms. The comparison results show the performance of parallel processing with the implemented GP-GPU is improved over ARM Cortex-A15 4Core, the most cutting-edge CPU in the embedded environment. The improvements are 17% in the integral image creation and 7% in 3x3 Gaussian filter mask.

Acknowledgements

This Research was supported by Seokyeong University in 2012.

References

[1] Advanced Micro Devices Inc, ATI CTM Guide, **(2006).**
[2] E. Lindholm, J. Nickolls, S. Oberman and J. Montrym, "NVIDIA Tesla: A Unified Graphics and Computing Architecture", Micro IEEE, vol. 28, **(2010),** pp. 39-55.
[3] J. Nickolls, I. Buck, K. Skadron and M. Garland, "Scalable Parallel Programming with CUDA", ACM Queue, vol. 6, **(2006)**, pp. 40-53.
[4] A. Levinthal and T. Porter, "Chap. A SIMD Graphics Processor", Computer Graphics, vol. 18, **(1984)**, pp. 77-82.
[5] R. A. Lorie and H. R. Strong Jr, "Method for conditional branch execution in SIMD vector processors", U.S Patent 4,435,758, **(1982)** August 13.
[6] J. Montrym and H. Moreton, "The GeForce 6800", Micro IEEE, vol. 25, **(2005)**, pp. 41-51.
[7] E. Lindholm, M. J. Kilgard and H. Moreton, "A user-programmable vertex engine", Proceedings of the ACM SIGGRAPH 2001, **(2001)** August 12-17; Los Angeles, USA.
[8] S. Liu, J. E. Lindholm, M. Y. Siu, B. W. Coon and S. F. Oberman, "Operand collector architecture", U.S. Patent 7,834,881, **(2006)** November 1.
[9] W. W. L. Fung, I. Sham, G. Yuan and T. M. Aamodt, "Dynamic Warp Formation and Scheduling for Efficient GPU Control Flow", Proceedings of the 40th Annual IEEE/ACM International Symposium, **(2007)** December 1-5; Chicago, USA.
[10] K. -y. Lee, N. -w. Eum and J. -c. Kwak, "Superscalar GP-GPU design of SIMT architecture for parallel processing in the embedded environment", Advanced Science and Technology Letters, vol. 43, **(2013)**, (Multimedia 2013), pp. 67-70.

Authors

Kwang-yeob Lee

He received a B.S. degree in the Department of Electronics Engineering at Seogang University in 1985. He received M.S. and Ph.D. degrees in the Department of Electronics Engineering at Yonsei University in 1987 and 1994. He is a professor in the Department of Computer Engineering at Seokyeong University since 1995. His research interests include Microprocessor, Embedded System, and 3D Graphics System.

Jae-chang Kwak

He received a B.A. degree at Yonsei University in 1983. He received M.S. and Ph.D. degrees in the Department of Computer Science at The University of Iowa in 1990 and 1993. He is a professor in the Department of Computer Science at Seokyeong University since 1995. His research interests include network protocols, QoS, network performance evaluation, and image processing algorithm

