Cluster Comput (2017) 20:781–787
DOI 10.1007/s10586-017-0768-9

A dynamic CTA scheduling scheme for massive parallel computing

Dong Oh Son[1] · Cong Thuan Do[1] · Hong Jun Choi[2] · Jiseung Nam[1] · Cheol Hong Kim[1]

Received: 27 October 2016 / Accepted: 30 January 2017 / Published online: 14 February 2017

Abstract Recent computing devices execute massive parallel data requiring huge computing hardware. To satisfy increasing computing need, GPUs providing powerful computational capability are employed to execute both graphics and general-purpose applications (GPGPUs). In the GPGPU, executing multiple applications together can increase the data parallelism, resulting in high resource utilization. Improving the resource utilization of the GPGPU can increase the GPGPU performance. However, various kinds of applications have different execution time depending on their workload sizes. Therefore, if one application is completed earlier than the other ones, resource underutilization problem may happen because the hardware resource allocated for the early completed application becomes idle. In this work, a CTA-aware dynamic streaming multiprocessors scheduling scheme is proposed for multiple applications execution in the GPGPU to efficiently manage hardware resources. Simulation results show that the proposed CTA-aware dynamic SM scheduling scheme can increase the GPU performance by up to 25.6% on average.

✉ Cheol Hong Kim
chkim22@jnu.ac.kr

Dong Oh Son
sdo1127@gmail.com

Cong Thuan Do
congthuan.hut@gmail.com

Hong Jun Choi
chj6083@nsr.re.kr

Jiseung Nam
jsnam@jnu.ac.kr

[1] School of Electronics and Computer Engineering, Chonnam National University, Gwangju, Korea

[2] The Attached Institute of ETRI, Daejeon, Korea

Keywords GPGPU · Multiple applications · SM scheduling scheme · Resource utilization

1 Introduction

In the recent computing systems market, the proportion of state-of-the-art embedded systems containing IoT, cloud computing, smart phone has been increased rapidly. Up-to-date embedded devices executing massive parallel data become more complex with increased size, whereas conventional embedded systems are composed of simple and cheaper hardware to meet the necessities of the target system. Especially, clouding systems operating heavy tasks require high parallel computing hardware. In this work, we focus on the embedded hardware for clouding systems where the performance is very important.

In order to enhance the throughput of computing systems, parallel hardware architecture to process massive parallel computing has been focused. To accomplish this, multicore architectures, which can provide better performance than single core architectures, have been proposed [1,2]. As the size of data processed by computing systems becomes bigger, the design cost of multicore architecture is increased dramatically to support massive parallel data [3]. Contrary to CPUs, GPUs mainly executing graphics applications can efficiently execute massive parallel data by employing powerful computation hardware with cheaper cost [4]. In general, the cost of CPU to compute hundreds TFLOPS is much higher than the GPU [5]. Moreover, GPU can execute general computing by exploiting new technology, leading to general purpose computation on graphic processing units (GPGPU) [6]. With the support of new parallel programing models such as CUDA [7], OpenCL [8], ATI Streaming [9], etc., GPGPUs are recognized more and more powerful in executing

general-purpose applications. Unfortunately, resource under-utilization problem may occur in GPGPUs, even though GPGPUs can concurrently process thousands of threads to increase data parallelism [5]. For this reason, many studies have focused on the techniques to enhance the parallelism of GPGPU [3,10–15].

In traditional GPUs, single application is executed at a time, because an application includes enough threads to fully utilize hardware resources on the GPU [16]. In recent GPUs such as Kepler architecture model GTX 780 Ti [17], hardware resources are growing with new generation. Therefore, most applications cannot fully utilize hardware resources in recent GPUs. In other words, one application cannot utilize maximum hardware resources on the GPU. Please note that GPUs generate kernels to execute applications. Therefore, we know that all GPU hardware resources are not fully utilized, if one kernel is executed [16]. To alleviate this problem, GPUs can support concurrent execution of multiple kernels, either from the same application or from different applications.

Execution time of an application depends on the workload size. In our previous work [18], we analyze the performance and resource utilization efficiency on GPGPU when multiple applications are executed concurrently compared to single application execution.

Based on the previous work, this paper proposes a CTA-aware dynamic streaming multiprocessors scheduling scheme to manage the hardware resources efficiently when multiple applications are executed concurrently.

The rest of this paper is organized as follows: Section 2 describes the GPU architecture, GPGPU application hierarchy and the Cooperative Thread Array. Section 3 describes the proposed CTA-aware Dynamic Streaming Multiprocessors Scheduling Scheme. Section 4 presents our experimental methodology and detailed results. Finally, Sect. 5 concludes this work.

2 Background

2.1 GPU architecture

In this section, we describe the GPU architecture and GPGPU application hierarchy briefly. In this work, we explain the target GPU architecture based on NVIDIA Fermi series [19]. GPU is a representative parallel architecture based on Single Instruction Multiple Data (SIMD). SIMD architecture has been widely used to maximize the hardware parallelism. Figure 1 shows the GPU architecture which models a similar to the NVIDIA Fermi architecture. A GPU consists of a number of GPU cores (or streaming multiprocessors SMs), with each SM having a SIMT width of 8–32 (Fermi architecture has 16 SMs with a SIMT width of 32). This figure depicts the general structure of Fermi architecture, which

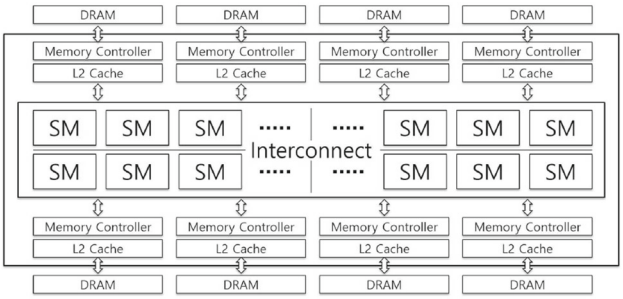

Fig. 1 GPU architecture

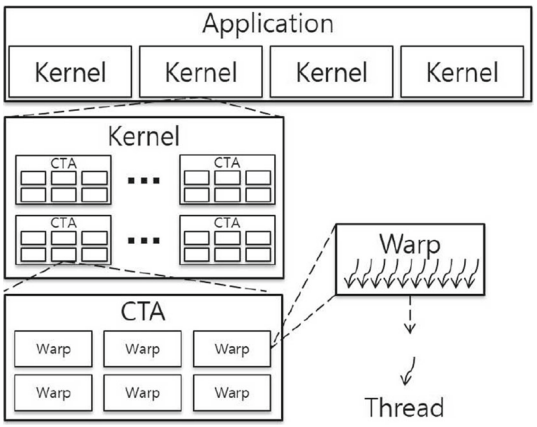

Fig. 2 GPGPU application hierarchy

consists of DRAM, L2 cache and streaming multiprocessors (SMs). SMs and DRAMs are connected via interconnection networks. Each SM consists of many streaming processors (SPs), a huge register file, shared memory, L1 cache, warp schedulers and dispatch units.

2.2 GPGPU application hierarchy

Figure 2 shows the hierarchy of a typical GPGPU application which consists of many kernels; each kernel is divided into many cooperative thread arrays (CTAs). Each CTA is sub-divided into groups of threads (called warps). The maximum number of threads per warp is usually 32 in modern GPGPU architectures. In the GPGPU architecture, GigaThread scheduler can schedule thread block in a round-robin fashion. Traditional GPUs execute single application at one time. In this paper, we focus on the recent GPU architecture where multiple applications are launched at the same time.

2.3 Cooperative thread array

In the conventional GPU, an application consists of many kernels and each kernel is divided into many CTAs which include many threads. In the architecture, only one single kernel can be executed at one time. CTAs are assigned to SMs

by a CTA scheduler—Giga Engine. CTA scheduler monitors the status of all CTAs within a CTA pool and then selects the CTA based on CTA scheduling schemes based on round robin fashion. Modern GPUs use Fine-Grained Multithreading (FGMT) which allows multiple CTAs to interleave their execution on one SM. This can increase the resource utilization and parallelism by hiding long memory access [20]. The number of CTAs assigning to one SM is limited by SM resource. In this work, the maximum number of CTAs assigning to one SM cannot exceeds 8. In the SM architecture, Single Instruction Multiple Threads (SIMT) execution model is employed, with the width of SIMT ranges from 8 to 32 that is equal to the warp size [4]. For NVIDIA GPU architecture, the warp size is 32. In this paper, our target GPU is modeled based on NVIDIA Fermi series.

3 CTA-aware dynamic streaming multiprocessors scheduling scheme

3.1 Multiple applications execution

In this section, we provide a detailed description for multiple applications execution in the GPGPU. Traditional GPUs execute only one single kernel at a time. However, recent GPUs can support concurrent execution of multiple kernels, either from the same application or from different applications. We assume that concurrently executing kernels are originated from multiple applications. Therefore, in this paper, application means the concurrently executing kernels. Figure 3 shows the conventional GPU architecture executing single application and multiple applications.

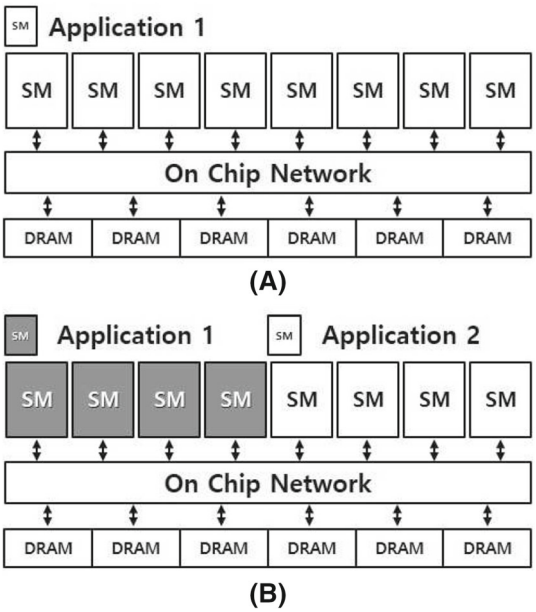

Fig. 3 Conventional GPU architecture executing, **a** single application, **b** multiple applications

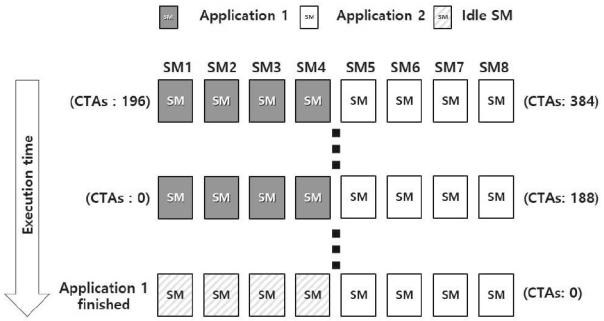

Fig. 4 Baseline scheme in multiple applications execution

3.2 CTA-aware dynamic streaming multiprocessors scheduling scheme

Figure 4 presents the example of multiple applications execution for the baseline scheme. As shown this figure, CTAs are assigned to SMs based on the SM scheduling scheme at the starting time. After then, SMs are running until all allocated CTAs are executed completely. If some SMs become idle (called idle SM), the idle SMs cannot be assigned to any CTAs. In the baseline GPU architecture, execution time of application varies due to different workload size. This causes underutilization of computational resources, resulting in GPU performance degradation.

To solve the problem, we propose a CTA-aware dynamic SM scheduling scheme that can reduce the allocable number of SMs of an application when the number of allocated CTAs is small (smaller application). Afterwards, CTA assignment from the smaller application is stopped. Several SMs previously assigned to the smaller application can be allocated to the other application after completing the remaining workloads. Therefore, this can increase the resource utilization. To illustrate the sequence of operations for the proposed CTA-aware dynamic SM scheduling scheme, we provide examples in multiple applications execution in Fig. 5.

In the GPGPU, execution time of CTA is different. To illustrate this, we assume that the execution time of CTA is equal. In Fig. 5, we assume that Application 1 has 196 CTAs and Application 2 has 384 CTAs.

(1) Each application is allocated 4 SMs at the starting time.
(2) Allocable SM of Application 1 is reduced when number of allocated CTAs is small (number of CTAs is 48). CTA assignment to the 4th SM is stopped. The 4th SM runs until it completes the remaining number of CTAs.
(3) Allocable SM of Application 1 is reduced when number of allocated CTAs is small (number of CTAs is 24) as shown above. CTA assignment to the 3rd SM is stopped.

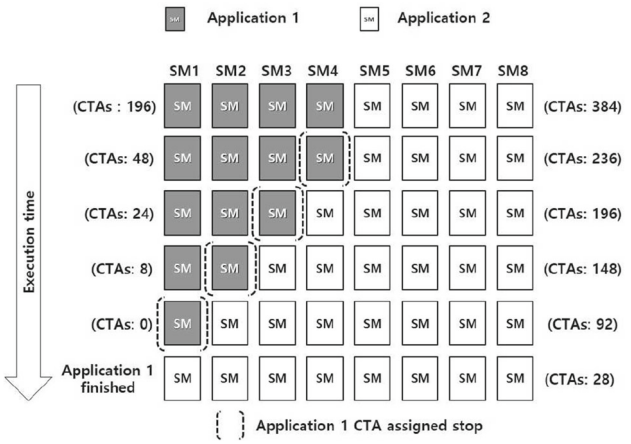

Fig. 5 Example of proposed scheme in multiple applications execution

Application 2 can assign the 4th SM when it completes all CTAs in the 4th SM.

(4) Application 1 reduces the number of SMs sequentially. To the contrary, Application 2 increases the number of SMs sequentially.

$$SM(n) = \sum_{k=1}^{n} k \tag{1}$$

In this work, we measure the number of assigned SMs using Eq. (1). This number is reduced when the number of CTAs of the application is equal to SM (n) where "*SM(n)*" is the number of remaining CTAs and "*CTAmax*" is the maximum number of CTAs that is limited by SM hardware resource.

4 Experimental methodology and results

4.1 Experimental methodology

In this section, we describe the details of our experimental methodology. To implement the proposed CTA-aware dynamic SM scheduling scheme, we modified the cycle-accurate simulator, GPUWattch [21]. This simulator is integrated GPU performance evaluation simulator GPGPU-SIM [22] and power measurement simulator McPAT [23]. We used the TeslaC2050 in NVIDIA Fermi architecture [19]. Table 1 presents the hardware parameters of the baseline GPU used in our experiments. In this simulation, maximum number of concurrent CTAs is 8. The benchmarks were selected from NVIDIA SDK [24] (see Table 2).

In this work, we mixed four workloads for simulation with short-term applications and long-term applications. Table 3 shows the configurations of mixed benchmark program using four benchmark programs.

Table 1 Hardware parameters

Parameter	Value
Number of SM	8
Warp size (SIMD width)	32
Number of threads/SM	1024
Shared memory/SM	48KB
Constant cache/SM	8KB, 2-way 64byte lines, read-only
Texture cache/SM	12KB, 24-way 128byte lines, read-only
L1 data cache	16KB, 4-way, 128byte lines
Unified L2 cache	64KB, 8-way, 128byte lines
Clock (core: interconnection: DRAM)	575MHz: 575MHz: 750MHz
Number of memory controller	8
Number of memory chip/controller	2
Memory channel bandwidth	4 bytes
GDDR3 memory timing	tCL=12, tRP =12, tRC=40,tRAS=28, tRCD=12, tRRD=6
Warp formation	post dominator
CTA&warp scheduler (scheduling scheme)	two-level scheduler (Round-Robin)

Table 2 Benchmark programs

Benchmark program	Abbreviation	CTAs	Execution time
Dct8x8	DC	4096	Short-term
SimpleTexture	ST	4096	Short-term
AsyncAPI	AA	32768	Long-term
BlackScholes	BS	4096	Long-term

Table 3 Workloads

Small size benchmark	Large size	Abbreviation benchmark
Dct8x8	AsyncAPI	DC-AA
Dct8x8	BlackScholes	DC-BS
SimpleTexture	AsyncAPI	ST-AA
SimpleTexture	BlackScholes	ST-BS

4.2 Experimental results

In this section, we present and discuss the simulation results of CTA-aware Dynamic SM Scheduling Scheme. Figure 6 shows GPGPU performance with CTA-aware dynamic SM scheduling compared to the baseline architecture. On average, CTA-aware dynamic SM scheduling scheme can improve the IPC of GPGPU by 25.6% without performance degradation in executing any benchmarks (50.4% for DC-

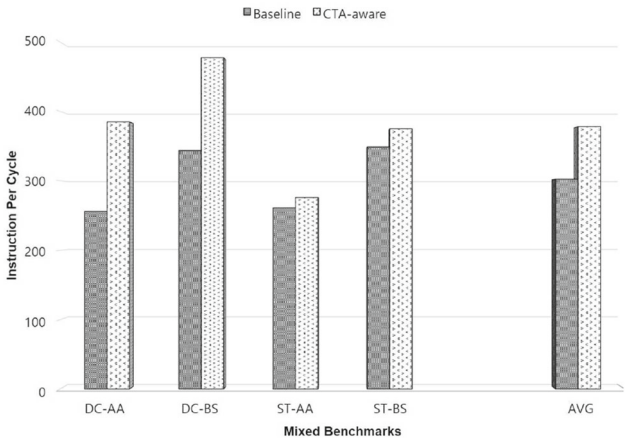

Fig. 6 IPC (instruction per cycle)

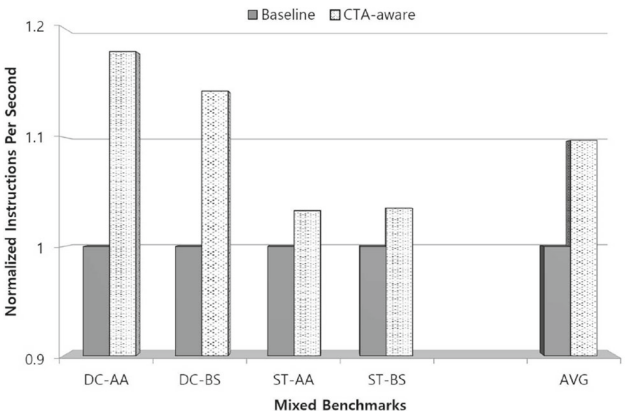

Fig. 7 IPS (instructions per second

AA, 39.0% for DC-BS, 5.6% for ST-AA, 7.5% for ST-BS). This demonstrates that the proposed CTA-aware dynamic SM scheduling scheme can provide higher resource utilization than the baseline architecture by executing multiple applications simultaneously.

Figure 7 shows the GPU instructions per Second according CTA-aware Dynamic SM Scheduling Scheme. Each bar in the graph is normalized to the instructions per second (IPS) of baseline GPU architecture. Note that the IPS is used to show the speed for instruction execution. Our simulation results show that the proposed CTA-aware dynamic SM scheduling scheme increases the IPS by 9.7% on average (17.8% for DC-AA, 14.2% for DC-BS, 3.3% for ST-AA, 3.5% for ST-BS).

According to our analysis, the CTA-aware dynamic SM scheduling scheme increases the hardware resource utilization by decreasing the proportion of idle SMs. The proposed scheduling scheme can improve the IPC and IPS when multiple applications are executed on the GPU simultaneously. Therefore, we expect that the proposed scheme can be a good solution for reducing the allocable number of SMs of an application in small-size application, leading to better overall GPU performance.

5 Conclusions

In this paper, we investigated the impact of multiple applications execution on GPGPU performance. GPGPU hardware resources can be utilized better by multiple applications execution. Therefore, the issue of resource utilization on executing multiple applications should be considered in designing recent GPGPUs. We proposed a new scheduling scheme which manages the streaming multiprocessors in the GPGPU more efficiently, resulting in improved resource utilization. Our proposed CTA-aware dynamic streaming multiprocessors scheduling scheme can manage the hardware resource of GPGPU considering the workload size of various applications when multiple applications are executed concurrently. According to our experimental results, our proposed scheduling scheme increased GPGPU hardware resource utilization significantly, resulting in 25.6% IPC (Instructions Per Cycle) improvement and 9.7% IPS (Instructions Per Second) improvement on average.

Acknowledgements This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2015R1D1A3A010 19454), and it was also supported by the MSIP(Ministry of Science, ICT and Future Planning), Korea, under the ITRC(Information Technology Research Center) support program (IITP-2016-R2718-16-0011) supervised by the IITP(Institute for Information & communications Technology Promotion).

References

1. Agarwal, V., Hrishikesh, M.S., Keckler, S.W., Burger, D.: CLK rate versus IPC: the end of the road for conventional microarchitectures, In: Proceedings of the 27th International Symposium on Computer Architecture. pp. 248–259 (2000)
2. Olukotun, K., Nayfeh, B.A., Hammond, L., Wilson, K., Chang, K.: The case for a single-chip multiprocessor. In: Proceedings of 7th Conference on Architectural Support for Programming Languages and Operating Systems, pp. 2–11 (1996)
3. Hill, M.D., Marty, M.R.: Amdahls law in the multicore era. IEEE Comput. **41**(7), 33–38 (2008)
4. Buck, I., Foley, T., Horn, D., Sugerman, J., Fatahalian, K., Houston, M., Hanrahan, P.: Brook for GPUs: stream computing on graphics hardware. In: Proceedings of Computer Graphics and Interactive Techniques (SIGGRAPH), pp. 777–786 (2004)
5. Lee, V.W., Kim, C.K., Chhugani, J., Deisher, M., Kim, D.H., Nguyen, A.D., Satish, N., Smelyanskiy, M., Chennupaty, S., Hammarlund, P., Singhal, R., Dubey, P.: Debunking the 100X GPU vs. CPU Myth: an Evaluation of Throughput Computing on CPU and GPU. In: Proceedings of International Symposium on Computer Architecture, pp. 451–460 (2010)
6. General-purpose computation on graphics hardware. http://www.gpgpu.org

7. NVIDIA CUDA programming. http://www.nvidia.com/object/cuda_home_new.html
8. OpenCL. http://www.khronos.org/opencl/
9. ATI Streaming. http://www.amd.com/stream
10. Tanasic, I., Gelado, I., Cabezas, J., Ramrez, A., Navarro, N., Valero, M.: Enabling preemptive multiprogramming on GPUs. In: Proceedings of the 41st International Symposium on Computer Architecture, pp. 193–204 (2014)
11. Xie, X., Liang, Y., Wang, Y., Sun, G., Wang, T.: Coordinated static and dynamic cache bypassing for GPUs. In: Proceedings of 21th IEEE International Symposium on High Performance Computer Architecture, pp. 76–88 (2015)
12. Voitsechov, D., Etsion, Y.: Single-graph multiple flows: energy efficient design alternative for GPGPUs. In: Proceedings of the 41st International Symposium on Computer Architecture, pp. 205–216 (2014)
13. Lee, S., Arunkumar, A., Wu, C.: CAWA: coordinated warp scheduling and cache prioritization for critical warp acceleration of GPGPU workloads. In: Proceedings of the 42st International Symposium on Computer Architecture, pp. 515–527 (2015)
14. Wu, G.Y., Greathouse, J.L., Lyashevsky, A., Jayasena, N., Chiou, D.: GPGPU performance and power estimation using machine learning. In: Proceedings of 21th IEEE International Symposium on High Performance Computer Architecture, pp. 564–576 (2015)
15. Lee, M., Song, S., Moon, J., Kim, J., Seo, W., Cho, Y., Ryu, S.: Improving GPGPU resources utilization through alternative thread block scheduling. In: Proceedings of 20th IEEE International Symposium on High Performance Computer Architecture, pp. 260–271 (2014)
16. Jog, A., Bolotin, E., Guz, Z., Parker, M., Keckler, S.W., Kandemir, M.T., Das, C. R.: Application-aware memory system for fair and efficient execution of concurrent GPGPU applications. In: proceedings of 7th Workshop on General Purpose Processing Using GPUs (2014)
17. NVIDIA GTX 780-Ti. http://www.nvidia.com/gtx-700-graphics-cards/gtx-780ti/
18. Son, D.O., Do, C.T., Choi, H.J., Kim, J.M., Park, J.H., Kim, C.H.: CTA-aware dynamic scheduling scheme for streaming multiprocessors in high-performance GPUs. In: Proceedings of Information Science and Applications (ICISA 2016), Vol. 376, pp. 1391–1399 (2016)
19. NVIDIAs Next Generation CUDA Compute Architecture: Fermi, www.nvidia.com/content/pdf/fermi_white_papers/nvidia_fermi_compute_architecture_whitepaper
20. Thornton, J.E.: Parallel operation in the control data 6600. In: Proceedings of AFIPS Proceedings of FJCC, Part. 2, Vol. 26, pp. 33–40 (1964)
21. Bakhoda, A., Yuan, G.L., Fung, W.W.L., Wong, H., Aamodt, T.M.: Analyzing CUDA workloads using a detailed GPU simulator. In: Proceedings of 9th International Symposium on Performance Analysis of Systems and Software, pp. 163–174 (2009)
22. Bakhoda, A.G., Yuan, L., Fung, W.W.L., Wong, H., Aamodt, T.M.: Analyzing CUDA workloads using a detailed GPU simulator. In: Proceedings of 9th International Symposium on Performance Analysis of Systems and Software, pp. 163–174 (2009)
23. Li, S., Ahn, J.H., Strong, R.D., Brockman, J.B., Tullsen, D.M., Jouppi, N.P.: McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In: Proceedings of the International Symposium on Microarchitecture, pp. 469–480 (2009)
24. CUDA SDK. http://developerdownload.nvidia.com/compute/cuda/sdk/website/samples.html

Dong Oh Son received the B.S. degree and M.S. in electronics and computer engineering from Chonnam National University, Gwangju, Korea in 2010 and 2012 respectively. He received the Ph.D. degree in computer engineering from Chonnam National University in 2016. Currently, he works as a postdoctoral researcher at Chonnam National University. His research interests include computer architecture, embedded systems and GPGPU.

Cong Thuan Do received the Engineer's degree from Hanoi University of Science and Technology, Hanoi, Vietnam in 2012, the M.S. degree in electrical and computer engineering from Chonnam National University, Gwangu, Korea in 2014. Currently, he is pursuing his Ph.D. in Electrical and Computer Engineering at Chonnam National University. His research interests include computer architecture, parallel processing, microprocessors, embedded systems and GPGPU.

Hong Jun Choi received the B.S. degree and M.S. in electronics and computer engineering from Chonnam National University, Gwangju, Korea in 2009 and 2011 respectively. He received the Ph.D. degree in computer engineering from Chonnam National University in 2014. From Sept 2014 to Jan 2015, he worked as a postdoctoral researcher at Chonnam National University. Currently, he works at the Attached Institute of ETRI. His research interests include computer architecture, low-power processors and GPGPU.

Jiseung Nam received the B.S. Degree in electrical engineering from Inha University, Korea, in 1981, the M.S. degree in electrical engineering from University of Alabama in 1986, and Ph.D. degree in electrical and computer engineering from University of Arizona in 1992. He worked as a researcher in Electronics and Telecommunications Research Institute (ETRI), Daejeon, Korea from 1992 to 1995. Currently, he is a professor in Chonnam National University, Gwangju, Korea. His research interests include Computer Network, IPTV, Smart TV, Multimedia Communication and Digital Media Arts.

Cheol Hong Kim received the B.S. degree in Computer Engineering from Seoul National University, Seoul, Korea in 1998 and M.S. degree in 2000. He received the Ph.D. in Electrical and Computer Engineering from Seoul National University in 2006. He worked as a senior engineer for SoC Laboratory in Samsung Electronics, Korea from Dec 2005 to Jan 2007. Now is working as an Associate Professor at School of Electronics and Computer Engineering, Chonnam National University, Korea. His research interests include embedded systems, mobile systems, computer architecture, SoC design, low power systems, and multiprocessors.