Bulletin of Electrical Engineering and Informatics
Vol. 10, No. 2, April 2021, pp. 917~926
ISSN: 2302-9285, DOI: 10.11591/eei.v10i2.2736

Architecture exploration of recent GPUs to analyze the efficiency of hardware resources

Viet Tan Vo[1], Cheol Hong Kim[2]
[1]School of Electronics and Computer Engineering, Chonnam National University, Gwangju, Korea
[2]School of Computer Science and Engineering, Soongsil University, Seoul, Korea

Article Info

Article history:

Received Aug 22, 2020
Revised Nov 12, 2020
Accepted Dec 6, 2020

Keywords:

Architecture
GPGPU
GPU
Multicore
Multiple warp schedulers
Performance

ABSTRACT

This study analyzes the efficiency of parallel computational applications with the adoption of recent graphics processing units (GPUs). We investigate the impacts of the additional resources of recent architecture on the popular benchmarks compared with previous architecture. Our simulation results demonstrate that Pascal GPU architecture improves the performance by 273% on average compared to old-fashioned Fermi architecture. To evaluate the performance improvement depending on specific hardware resources, we divide the hardware resources into two types: computing and memory resources. Computing resources have bigger impact on performance improvement than memory resources in most of benchmarks. For hotspot and B+ tree, the architecture adopting only enhanced computing resources can achieve similar performance gains of the architecture adopting both computing and memory resources. We also evaluate the influence of the number of warp schedulers in the SM (streaming multiprocessor) to the GPU performance in relationship with barrier waiting time. Based on these analyses, we propose the development direction for the future generation of GPUs.

Corresponding Author:

Cheol Hong Kim
School of Computer Science and Engineering, Soongsil University
369 Sangdo-Ro, Dongjak-Gu, Seoul 06978, South Korea
Email: cheolhong@ssu.ac.kr

1. INTRODUCTION

Multithreaded hardware, which has been extensively developed in the last decade, becomes an attractive computing unit for software developers to speed up their solution of computational problems. Graphics processing units (GPUs) form one of the typical classes of multithreaded hardware. Modern GPU generation provides parallel programing models such as CUDA [1] and OpenCL [2, 3], which make GPUs accessible for non-graphics applications as well as graphics applications. The increasing demand from developers for more powerful GPUs along with flexible development platforms requires GPU manufactures to pack more memory and execution units with higher speeds, as well as improve their programmability. Most recent GPU architectures have adapted to this trend, however, hardware resources in GPUs are not fully utilized yet, leading to reduced performance improvement.

A typical GPU comprises multiple streaming multiprocessors (SMs), or compute units (CUs) as commonly referred by AMD, also known as shader cores [4, 5]. SMs are layouted in groups called clusters that share a common port to the interconnection network. Each SM contains a certain amount of single-instruction, multiple-thread (SIMT) cores or CUDA cores, several load/store units (LSUs), and special function units (SFUs). The core employs integer arithmetic logic units (ALUs) and floating-point units (FPUs) supporting diverse instructions including boolean, shift, move, compare, convert, bit-field extract, bit-reverse insert, and

population count. The LSU is responsible for memory operations, whereas SFU is dedicated for transcendental instructions such as sine, cosine, reciprocal, and square root [6-8].

A general-purpose computing on GPUs (GPGPU) application comprises several kernels, each comprising a massive number of threads (instance of the program kernel). A group of threads initiates a thread block termed as a cooperative thread array (CTA) [6, 9]. The size of a CTA is determined by the number of threads that the programmer intends to launch at a time. Determing the size of CTA wisely allows a balanced workload distribution to SMs, which enables utilizing hardware resources efficiently. Global CTA scheduler assigns one or more CTAs to the SM until the resources inside the SM are saturated. A new CTA is assigned to the SM whenever it completes the previous CTA. Within the SM, CUDA cores and other execution units execute the threads in groups of 32 threads known as warps. The warps stored in a warp pool in the SM are scheduled for their execution in the order with respect to the policy governed by the warp scheduler. On every cycle, the scheduler selects one of ready warps to issue next. Beside conventional loose round robbin (LRR) warp scheduler, there has been several efforts to effectively arrange the oder of warps such as greendy then oldest (GTO) [10], Two-level warp scheduler [11], Cache-consicous wavefront scheduling [10], Criticality-aware warp scheduling [12], and Synchronize-aware warp scheduling [13]. In modern GPU architectures, two or more warp schedulers are working together in the SM to better utilize the hardware resources.

To facilitate memory pipeline state, there are a few levels of memory units in the SM. A substantial portion of the register file is utilized for each SM. L1 caches help handle register spills and serve various purposes (e.g. read-only constant cache, texture data, and irregular data access) [6, 14-16]. Ranking at a similar hierarchy of the L1 cache, shared memory allows threads to cooperate with each other, which is the key feature for reusing on-chip data. Shared memory also helps to reduce the traffic to lower level caches. The applications that make use of shared memory often desire to perform barrier synchronization which is supported by CUDA. The use of synchronization barrier is to synchronize shared data between threads within a CTA [17]. Barriers are also used for correctness and maintaining the performance after a highly branching threads. When the warps of CTA hit the barrier, they become stalled and have to wait for all the other warps to arrive at the barrier before making any further progress. Back to the memory pipeline state, if the memory requests are out of the L1 range, they are recorded by a miss status holding registers (MSHRs) before proceeding to L2 cache via an interconnect network. The L2 cache can be accessed across the entire kernel. It is split into multiple banks attached to the memory controllers. The controller is responsible for forwarding the miss requests in the L2 cache to the dynamic random access memory (DRAM) module. Figure 1 represents the general placement of the internal hardware components in recent GPUs.

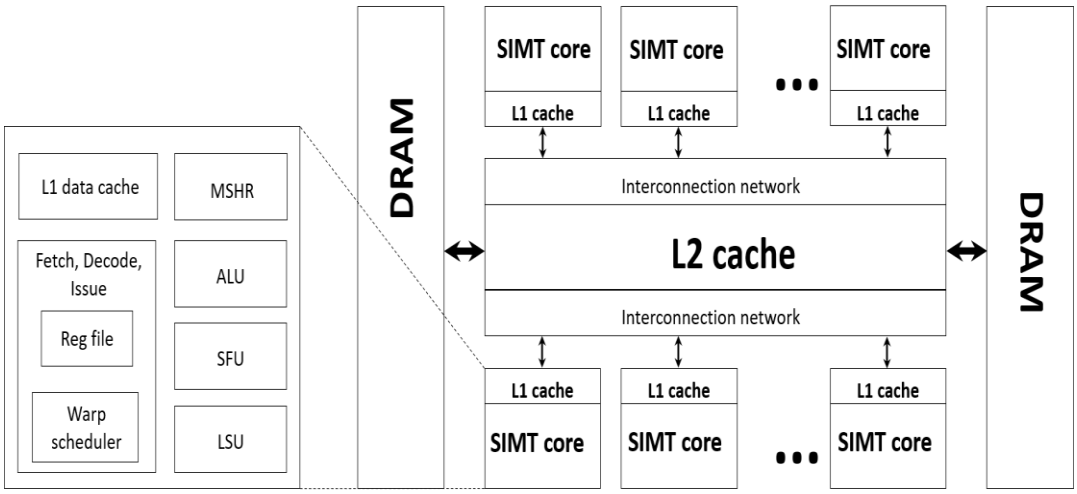

Figure 1. Baseline GPU architecture

There has been several ideas and techniques for improving the performance of GPUs. However, most of them are obtained from a specific GPU architecture therefore the compatibility with various generations is questionable. In fact, whenever a new GPU architecture is released, studies are conducted to evaluate the cability and the power of new GPUs. Although these studies do mentiton the advancement compared to the previous architecture, there is lack of a follow-up from generation to generation to measure how far the

architecture has developed. This is what our study tries to analyze. We not only evaluate the performance improvement over the predecessor architectures by applying the benchmarks but also analyze the efficiency of the hardware upgrades from fromer Fermi architecture to recent Pascal architecture in storage and computation domain. The understanding of the hardware efficiency throughout many GPU architectures is important for hardware developer to improve the future GPU architecture and for software developer to customize their applications to be optimized on various GPU generations. The remainder of this paper is organized as follows: Section 2 presents a comparison of the main improvements of Pascal to those of Fermi architecture. Section 3 presents a discussion of our experimental method and Section 4 presents the simulation results. Finally, Section 5 provides conclusions.

2. OLD ARCHITECTURE VERSUS RECENT ARCHITECTURE

Table 1 lists out some key improvements over the GPU development period of NVIDIA.

Table 1. GPU architecture comparison

Architecture	GT200 *(Tesla)*	GF110 *(Fermi)*	GM200 *(Maxwell)*	GP100 *(Pascal)*
GPU name	GTX 280	GTX 580	Tesla M40	Tesla P100
Transistors - process	1.4 bilion - 65 nm	3.0 bilion - 40 nm	8.0 bilion – 28nm	15.3 bilion – 16nm
CUDA Cores	240	512	3,072	3,584
SM	30	16	24	56
Base:Boost clock (MHz)	602:-	772:-	948:1,114	1,328:1,480
FP32 performance	622.1 GFLOPs	1.581 TFLOPs	6.844 TFLOPs	10.61 TFLOPs
FP64 performance	77.76 GFLOPs	197.6 GFLOPs	213.9 GFLOPs	5.304 TFLOPs
Shared memory/SM (KB)	16	16 or 48	96	64
L1 cache/SM (KB)	None	48 or 16	48	24
L2 cache (KB)	None	768	3,072	4,096
Memory interface	512-bit GDDR3	384-bit GDDR5	384-bit GDDR5	4096-bit HBM2
Memmory clock (MHz)	1,107	1,002	1,502	715
Bandwidth (GB/s)	141.7	192.4	288.4	732.2

Major revision of Tesla architecture (G80 and GT200) has been to set the philosophy at NVIDIA for improving both capability and programmability. The Fermi architecture has maintained this orientation and advanced further. NVIDIA learned from prior processors and improved several features to enhance GPU computation capability as:

a. Third generation SM [6]: a fourfold increase in the number of CUDA core (32 compared to 8 in GT200), 16 load/store units that set the address calculation speed at 16 threads per clock, four SFUs per SM, integer ALU in Fermi which supports full 32-bit precision for all instructions (GT200 is limited to 24-bit precision for multiplication operations), dual warp schedulers which enable two warps to be issued and executed simultaneously.

b. Memory subsystem improvement [6]: a true cache hierarchy with configurable L1/shared memory and unified L2 cache. The configurability allows programmers to configure the program behavior to fit their needs. Fermi was also the first GPU generation with error correcting code support from register files to DRAM.

Since the great success of Fermi, NVIDIA further improved their architecture for the next generations: Kepler and Maxwell. Pascal maintained this trend and became the most advanced NVIDIA architecture ever built [18]. Among the Pascal GPUs, GP100 is very powerful and architecturally complex GPU. Some of its key features are as:

a. NVLink provides high-speed bidirectional interface for GPU-to-GPU transfer. Multiple-GPU systems substantially benefit from this feature.

b. The GPU which utilizes the second generation of HBM (HBM2). This type of memory achieves a significant bandwidth upscale.

c. Sixth generation SM: a full GP100 delivers a total of 60 SMs, and each SM comprises 64 single precision (FP32) CUDA cores, 32 double precision (FP64) CUDA cores, and four texture units. Notably, FP32 CUDA cores can process both 16-bit and 32-bit precision instructions and data. This feature significantly benefits deep learning algorithms wherein high levels of precision are not strictly required. FP16 reduces memory usage, thus allowing larger training networks.

While Fermi and Kepler allow a programmer to configure shared memory and L1 cache, Maxwell and Pascal adopt a different approach. Each SM has its own 64 KB dedicated shared memory. Moreover, L1 cache can provide the function as a texture cache depending on the workload. Pascal GP100 also supports big L2 cache, which reduces DRAM access latency.

3. EXPERIMENTAL METHOD

In oder to evaluate how well the recent GPU architectures convert their hardware upgrades into performance gains, we use 9 applications (HS: Hotspot, BFS: Breadth-first search, LUD: LU decomposition, BP: BackPropagation, Dwt2d: discrete wavelet transfrom 2D, BT: B+ tree, PF: PathFinder, NW: Neadleman-Wunsch, SR: Srad_v2) from Rodinia benchmark suite [19] on a cycle level GPU simulator called GPGPU-Sim [20]. The simulator models the whole GPU allowing researcher to modify the architecture and run GPGPU applications without using a real GPU. To the best of our knowledge, almost all other researchers in GPU community have used GPGPU-Sim (v3.x) for their evaluation because Fermi architecture, which is supported by this version, is one of the simplest architecture configurations. However, we use GPGPU-Sim version 4.0 [21] since this version implements many updates to improve the evaluation accuracy and it supports many recent GPU architectures. Rodinia benchmark suite is a collection of popular benchmarks written in CUDA and OpenCL for evaluating the power and efficiency of GPU architecture.

We select popular Fermi GPU architecture to compare with recent Pascal GPU architecture. We apply the default architecture configuration of GTX 480 (Fermi) because it is the commonly used in the research community. For the case of recent architectures, GPGPU-Sim 4.0 supports Titan X configuration as a representative in Pascal family. Although TitanX does not use the most powerful Pascal chipset, it can generate a comparable performance as it uses GP102 chipset which inherits most of GP100 features excluding FP64 CUDA cores and replaces HBM2 memory interface by GDDR5X. Titan X is also more practical because it is a commercial GPU while GP100 is mainly available in high performance computer systems for scientific projects. We modified some parameters of Titan X to sufficiently approximate the real GPU. Table 2 describes the simulation parameters for compared two architectures. We divide the parameter set into two groups: core related parameters and memory related parameters.

Table 2. Simulation parameters

	Components	GTX 480	TITAN X
Core parameters	# of processing cores	15	56
	Clock (core:interconnect:L2:Dram)	700:700:700:924	1417:1417:1417:2500
	# of registers per core	32768	65536
	Max. # of warp per SM	48	64
	Max. # of CTA per SM	8	32
	Shared memory (KB)	48	48
Memory parameters	L1 data/instruction cache	16 KB/2KB 4-way	24 KB/4KB 48-way
	L2 cache	768 KB	3 MB
	# memory controlers (Dram channels)	6	12
	# of warp scheduler per SM	2 GTO schedulers	2 GTO schedulers

One prominent feature in modern GPU architectures is the hardware resource management by control the number of warps issued to executing units. The SM consists of a large number of CUDA cores that require an effective number of warp schedulers to orchestrate well for the activities of the warps. Fermi employs dual warp schedulers while Kepler and Maxwell use quad warp schedulers to utilize massive number of CUDA cores in their architecture. Recent GPU architecture like Pascal adopt dual warp schedulers as it is not necessary to use more than two warp schedulers because Pascal architecture has reduced the number of CUDA cores while it has increased the number of SMs. The key fact is that the number of warp schedulers per SM should be maintained appropriately depending on the hardware resources in the SM. If the number of warp schedulers is small while available CUDA cores are too many, most of CUDA cores are left to be unused. In the opposite case, the redundant warp schedulers may harm the GPU performance. To analyze the efficiency of hardware resources in the SM, we vary the number of warp schedulers in the SM. The number of warp schedulers does not only reflect the ability to utilize SM's resources but also affect to the barrier waiting time of the warps at the synchronization barrier. The fact is that synchronization overhead can be a limitation to achieve good performance in the GPU [19]. That is the reason why we also analyze barrier synchronization impact in overall performance. In this papper, we focus on inter-CTA synchronization which is entirely managed by hardware [22, 23]. Another type of synchronization is global synchronization which is achieved by allowing the current kernel to complete and start a new kernel or atomic operations [13], which cost a significant overhead compared to inter-CTA synchronization. CUDA provides *syncthreads()* to perform a barrier statement [24].

To measure the barrier waiting time, we trigger the cycle counter for every warp within a CTA when it hits the barrier and keep tracking until the final warp of that CTA reach the barrier. Since then, all warps are released and cross that synchronization barrier. The average waiting time for every CTA and SM are calculated for individual benchmark. We compare the waiting duration in case of single, dual, and quad warp schedulers for Fermi and Pascal architecture from most active synchronization kernels of each benchmark.

4. EVALUATION RESULTS

Table 3 presents the performance of the benchmarks running on Fermi and Pascal architectures which is normalized to Fermi's performance. As shown in the table, all the benchmarks show better performance in the modern Pascal architecture (273% performance improvement on average) . However, massive upgrade of hardware in the Pascal does not consistently translate into significant performance gains for every benchmark. The understanding of this inefficiency is necessary for architecture researchers to propose any development for the future GPU architecture generation. Based on our simulation results, we can know that the improvement divergence depends heavily on the characteristic of applications.

Hotspot (HS) is a relatively compute-intensive benchmark [19]. Therefore, the increase in the number of SMs and CUDA cores of Pascal is dominantly beneficial to the performance. BT (B+ tree) comprises several parallel regions (a region between two consecutive barrier instructions) [12]. It also shows over 4 times speed up in the Pascal over the Fermi owing to the improvement in parallelism ability of modern GPU architectures. LUD (LU decomposition) shows similar instructions per cycle (IPC) in both architectures. LUD algorithm, which involves matrices, exhibits significant inter-thread sharing as well as row and column dependencies [25]. LUD would strongly benefit from shared memory. However, shared memory still remained even in recent architecture compared to past architecture as 48 KB, which explains why the GPU performance remains unchanged in both architectures for LUD.

To gain further insight into the influence of the computing and memory parameters in recent GPU architectures, we define two parameter sets. By maintaining core related parameters as the Ferimi configuration then replacing the memory parameters from the Pascal architecture, which is named FermiCore_PascalMem, we first evaluate the benefits of the Pascal's memory related resources. Via the inverse procedure which is named PascalCore_FermiMem, we can analyze the contribution of computing resources in Pascal architecture. Figure 2 illustrates the performance comparison according to the improvement of memory and computing resources from Fermi to Pascal.

Table 3. Pascal IPC compared to Fermi IPC

Benchmarks	Fermi IPC	Pascal IPC	Pascal IPC Normalized to Fermi IPC
HS	605.9598	2601.3569	4.29
LUD	34.3512	34.6146	1.01
BP	548.5601	1600.2274	2.92
Dwt2D	219.5495	423.7977	1.93
BT	399.6233	1675.475	4.19
PF	701.1052	1966.9971	2.81
NW	22.1738	29.537	1.33
BFS	61.9692	197.1128	3.18
SR	540.6367	1585.5934	2.93

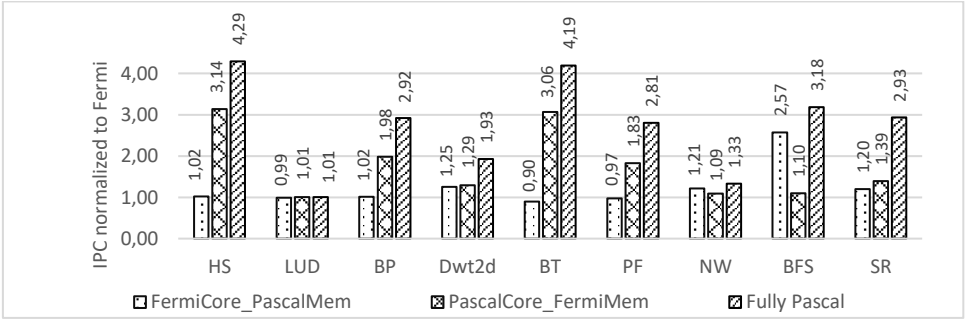

Figure 2. Performance comparison normalized to Fermi

As shown in Figure 2, HS and BT are two benchmarks that mostly inherit the benefit of core related parameters from recent GPU architecture (273% performance improvement on average). These core related configurations help to push the performance more than 3 times compared to Fermi performance. Digging into the simulation results of these benchmarks, when Pascal's core related parameters are applied, HS can schedule 4 more CTAs into one SM and BT can schedule 3 more CTAs, resulting in enhanced parallelism. These two benchmarks benefit from doubling the number of registers localized per SM in Pascal architecture. On the other hand, NW (Needleman-Wunsch) and BFS (Breadth-First Search) aggressively make use of memory related parameters of the new architecture. BFS in Pascal memory architecture achieves 2.57 times more efficient than

Fermi. In case of NW, Figure 2 shows that there is an improvement of 1.21 times contributed by memory parameters compared to 1.33 times of fully Pascal's parameters. NW has only 16 threads (corresponding to 1 warp) in a CTA [19], which makes it difficult to gain a big improvement with the new architecture because SM occupancy is throttled by a small number of threads in the CTA, although smaller CTAs and reduced occupancy can have gains in some cases. To conduct futher analysis on the benchmark characteristics as well as to find out effective hardware parameteres, we divide the benchmarks into four groups as:

Group 1: As shown in Figure 2, Dwt2d and SR show minimum dependency on the increase of computing and memory resources. When varying one of both types of resources, there is a little performance gain contributed by each type of resources. In both benchmarks, core related parameters have bigger impact than memory related parameters. The interesting thing is applying fully Pascal parameters provides an obvious performance improvement by 1.93 times and 2.93 times faster for Dwt2d and SR, respectively. These benchmarks are well-programmed to be scalable with the additional resources offered by recent architecture. For example, SR is a relatively compute-intensive benchmark, which uses matrix-like structure and exposes massive data parallelism. It also requires significant CPU-GPU communication which needs high memory bandwidth to facilitate [19]. Those reasons explain why SR shows noticeable performance improvement by increasing both computing and memory resources.

Group 2: There exists only LUD benchmark in this group because it shows no performance improvement in the new GPU architecture. As aforementioned, this benchmark mostly relies on the ability of shared memory utilization. While the amount of shared memory is unchanged in Pascal architecture, increasing memory channels and cache size in Pascal does not show any advantage for LUD.

Group 3: HS(Hotspot), BP(Backpropagation), BT(B+tree), and PF(PathFinder) are those benchmarks whose performance strongly depends on computing resources. In order to analyze the effects of computing resources, we evaluate the GPU performance by changing the number of processing cores and monitoring how the benchmarks react. Figure 3 indicates that the performance gain is 222% on average if we just double the number of processing cores. In detail, the performance of HS and BT are proportional to the number of shader cores. There is a spontaneous improvement (approximately 100%) when the number of cores increases from 16 to 36. The normalized IPC in these two benchmarks keep increasing as more cores are appended. However, the increasing slew rate is not sharp. Based on these results, we can know that there is still some scope for further performance improvement in the future GPU architecture. In case of BP (Backpropagation) benchmark, the IPC drops slightly when the number of cores reaches 76. This implies that the BP benchmark achieves its peak performance before the number of cores is increased to 76. PF (PathFinder) shows an abnormal behavior. The IPC does not consistently improve as more cores are added. One of the reasons is that PF is a high branch divergent benchmark, therefore, under-utilization of computing resources may occur.

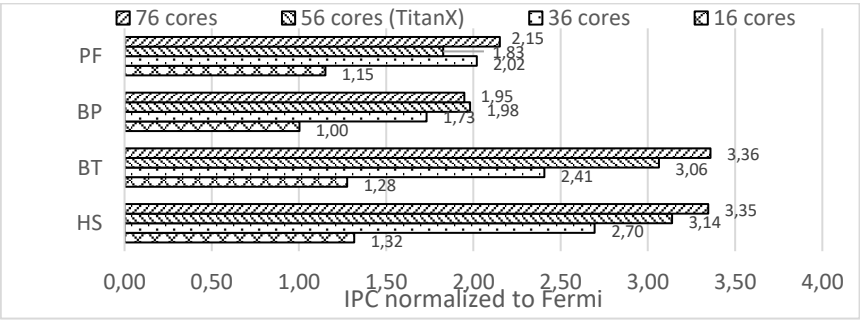

Figure 3. Performance impact of computing resources

Group 4: Only NW (Needleman-Wunsch) and BFS (Breadth-First Search) show similar bahavior in this group, which is highly influenced by memory resources. We maintain the computing resources similar to Fermi in combination with Pascal memory's configuration while varying the number of memory channels (memory controllers). Increasing the number of memory channels is more practical than increasing the size of cache. Figure 4 presents our simulation results for this experiment. The performance is increased by 206% if we increase the number of memory channels by 3 times. Both BFS and NW benchmarks are limited by the GPU off-chip bandwidth. However, only BFS is responsive when the number of memory channels increases. Meanwhile, NW shows an unconventional memory access pattern, and is designed for extensive use of shared memory. Therefore, adding more memory channels is not an effective way to improve the performance for this kind benchmarks.

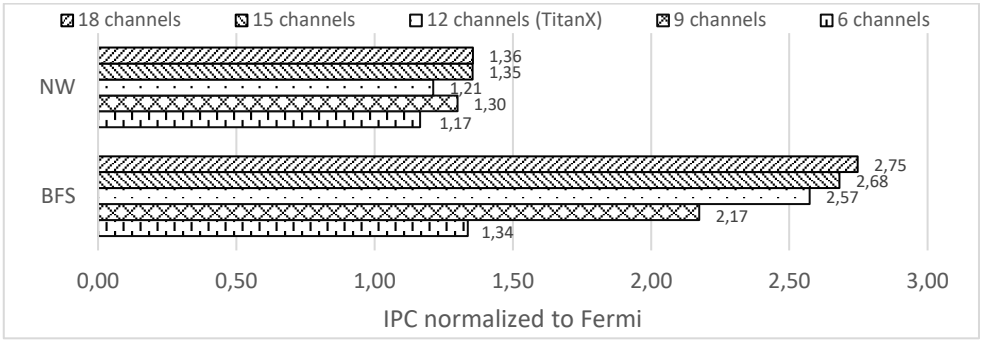

Figure 4. Performance impact of memory resources

In the following evaluation, we observe the performance difference while changing the number of warp schedulers in the SM. We also measure the barrier waiting time for evaluated cases. The number of warp schedulers play a critical role in maintaining the activities of all the memory and execution units within the SM. Multiple warp schedulers can issue many warps into the execution pipeline, leading to reduced waiting time in the warp pool. In some cases, this also increases the possibility that remaining warp within one CTA hit and clear the barrier synchronization faster. In other words, multiple warp schedulers encourage the CTA to finish earlier by enabling more CTAs to be lanched. Hence, it is necessary to evaluate the impact of the number of warp schedulers along with the barrier waiting time.

Figure 5 shows the IPC varying the number of warp schedulers in Fermi GPU architecture. The warp scheduler selects a ready warp and issues one instruction from it to a group of 16 CUDA cores, 16 load and store units or a group of four SFUs [6]. The warp schedulers only issue the instructions with respect to the avalabilty of hardware resources. Those benchmarks which are composed of a combination of memory and computational instructions have high probability to issue more warps to SM's resources. Our experimental results show that four benchmarks (HS, BP, PF, SR) out of nine benchmarks are obviously empowered by increasing the number of warp schedulers. HS and SR are compute intensive benchmarks, therefore they can utilize two groups of 16 CUDA cores owing to the help of multiple warp schedulers. BT is a memory intensive benchmark, which does not make use of multiple schedulers because there is only one group of 16 load/store units in the SM. Similarly, NW and BFS are categorized as memory related benchmarks where similar pattern is provided. Figure 6 shows the IPC with different number of warp schedulers in Pascal architecture. BT, Dwt2d, and LUD do not show IPC difference when dual or quad warp schedulers are applied, where more time is spent in waiting at the synchronization barrier. In contrast, HS and BP fully take advantage of adopting multiple warp schedulers, resulting in the performance improvement (more than 1.7 times) compared to those with single warp scheduler.

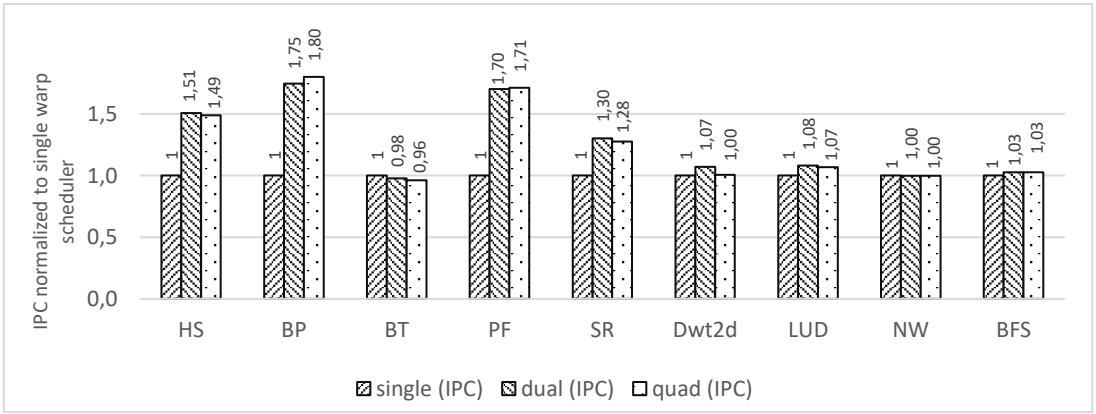

Figure 5. Impact of the number of warp schedulers to the GPU performance in Fermi architecture

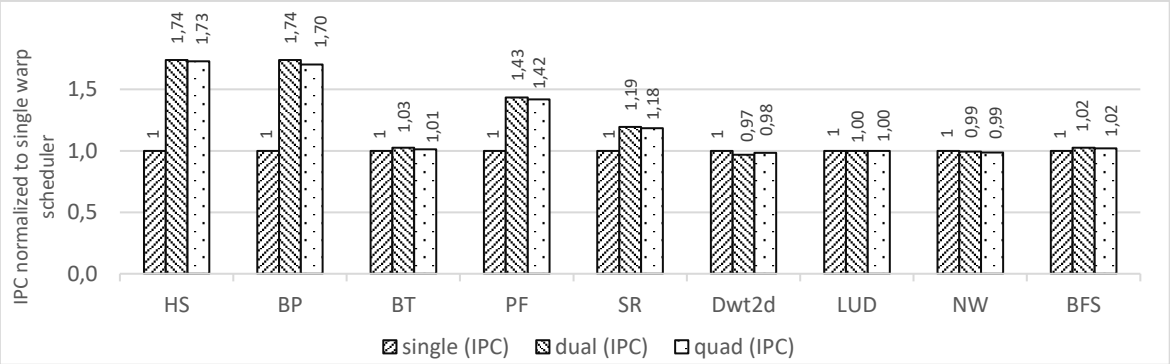

Figure 6. Impact of the number of warp schedulers to the GPU performance in Pascal architecture

Table 4 shows the barrier waiting time of multiple (dual and quad) warp schedulers normalized to the barrier waiting time of single warp scheduler in Fermi and Pascal archtectures. In this table, barrier waiting time for NW and BFS benchmarks are not provided. The reason is NW has only one warp (16 threads) in a CTA. Hence, warp does not have to wait for any other warps to arrive and cross the barrier in NW. In case of BFS, we cannot measure the barrier waiting time because there is no synchronization instruction in the benchmark. For BT, the barrier waiting time of multiple warp schedulers is longer than that of single warp scheduler, becuase synchronization instructions are performed frequently. This means that multiple warp schedulers can increase the barrier waiting time, leading to reduced performance. In the benchmarks where multiple warp schedulers show shorter barrier waiting time than that of single warp scheduler, multiple warp schedulers can improve the GPU performance more efficiently. From the results in Table 4, we also can know that using four warp schedulers is not always better than using two warp schedulers where two warp schedulers are enough to utilize the hardware resources in the SM. Therefore, the number of warp schedulers in the SM should be determined after careful evaluation about hardware utilization and barrier waiting time.

Table 4. Barrier waiting time depending on the number of warp schedulers

	Fermi		Pascal	
Benchmarks	Dual warp schedulers	Quad warp schedulers	Dual warp schedulers	Quad warp schedulers
HS	0.43	0.40	0.75	0.97
BP	0.95	0.73	0.92	0.75
BT	1.11	1.14	1.09	1.10
PF	0.48	0.42	0.73	0.59
SR	0.62	0.62	0.67	0.67
Dwt2d	0.75	0.82	1.13	1.16
LUD	0.82	0.88	0.99	1.14

5. CONCLUSION

In this paper, we explored the advantage of recent GPU architectures over former architectures with respect to different computing resources and memory resources. We focused on the performance analysis with two representative GPU architectures: Fermi for the old architecture and Pascal for the modern architecture. When the benchmarks are executed on Pascal architecture, the performance is improved by 273% on average and up to 429% for hotspot compared to Fermi architecture. In order to analyze the efficiency of hardware resources in the GPU, we conducted various simulations to evaluate the performance improvement with many combinations of computing and memory parameters such as the number of shader cores, the number of memory channels, and the number of warp schedulers. Our simulation results indicate that shader cores and memory channels are critical factors to improve the performance of modern GPU architecture. We observed that bandwidth consuming benchmarks can improve the performance by 206% if the number of memory channels is increased by three times, and computation sensitive benchmarks show the speedup of 222% on average if the number of cores is doubled. For most of evaluated benchmarks, utilizing computing and memory hardware efficiently provides noticeable speedup. The experiments also pointed out some cases where applying new architecture does not improve the performance. For this reason, enhancing the hardware resources in the GPU should be considered together with the characteristics of the applications. We also analyzed the efficiency of employing multiple warp schedulers in the SM. Our experiments showed that the number of warp schedulers

in the SM is well determined in recent GPU architectures, where multiple warp schedulers increase the hardware utilization and reduce the barrier waiting time for synchronization. However, excessive warp schedulers may cause resource under-utilization, leading to reduced performance. Therefore, appending more warp schedulers need to be taken into accout carefully if the future GPU architecture increases the hardware resources in the SM to solve more complex problems.

ACKNOWLEDGEMENTS

This work was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2018R1A2B6005740).

REFERENCES

[1] NVIDIA, "CUDA C++ Programing Guide," pp. 1-335, 2012.
[2] Khronos Group, "The OpenCL C Specification Version: 2.0," pp. 1-99, 2013.
[3] W. Q. Yan, "Introduction to Intellegent Surveillance," *SpringerNature*, 2016.
[4] C. T. Do, J. M. Kim, and C. H. Kim, "Application Characteristics-Aware Sporadic Cache Bypassing for high performance GPGPUs," *Journal of Parallel and Distributed Computing*, vol. 122, pp. 238-250, 2018, doi: https://doi.org/10.1016/j.jpdc.2018.09.001.
[5] D. O. Son, C. T. Do, H. J. Choi, J. Nam, and C. H. Kim, "A dynamic CTA scheduling scheme for massive parallel computing," *Cluster Computing*, vol. 20, no. 1, pp. 781-787, 2017, doi: 10.1007/s10586-017-0768-9.
[6] Devyani Tanna, "NVIDIA's Next Generation CUDA Compute Architecture: Fermi," 2009.
[7] H. J. Choi, D. O. Son, and C. H. Kim, "Memory Contention Aware Power Management for High Performance GPUs," in *Parallel and Distributed Computing, Applications and Technologies*, pp. 220-229, 2019.
[8] C. T. Do, J. M. Kim, and C. H. Kim, "Early miss prediction based periodic cache bypassing for high performance GPUs," *Microprocessors and Microsystems*, vol. 55, pp. 44-54, 2017. doi: https://doi.org/10.1016/j.micpro.2017.09.007.
[9] X. Chen, L. Chang, C. I. Rodrigues, J. Lv, Z. Wang and W. Hwu, "Adaptive Cache Management for Energy-Efficient GPU Computing," *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, Cambridge, pp. 343-355. 2014. doi: 10.1109/MICRO.2014.11.
[10] T. G. Rogers, M. OConnor and T. M. Aamodt, "Cache-Conscious Wavefront Scheduling," *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, Vancouver, BC, pp. 72-83, 2012. doi: 10.1109/MICRO.2012.16
[11] V. Narasiman, M. Shebanow, C. J. Lee, R. Miftakhutdinov, O. Mutlu, and Y. N. Patt, "Improving GPU performance via large warps and two-level warp scheduling," *MICRO-44: Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 308-317, 2011, doi: 10.1145/2155620.2155656.
[12] S. Lee and C. Wu, "CAWS: Criticality-aware warp scheduling for GPGPU workloads," *2014 23rd International Conference on Parallel Architecture and Compilation Techniques (PACT)*, Edmonton, AB, pp. 175-186, 2014. doi: 10.1145/2628071.2628107.
[13] J. Liu, J. Yang and R. Melhem, "SAWS: Synchronization aware GPGPU warp scheduling for multiple independent warp schedulers," *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Waikiki, HI, pp. 383-394, 2015. doi: 10.1145/2830772.2830822
[14] G. B. Kim, J. M. Kim, and C. H. Kim, "Dynamic Selective Warp Scheduling for GPUs Using L1 Data Cache Locality Information," in *Parallel and Distributed Computing, Applications and Technologies*, pp. 230-239, 2019.
[15] C. Nugteren, G. van den Braak, H. Corporaal and H. Bal, "A detailed GPU cache model based on reuse distance theory," *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, Orlando, FL, pp. 37-48, 2014. doi: 10.1109/HPCA.2014.6835955.
[16] H. Wong, M. Papadopoulou, M. Sadooghi-Alvandi and A. Moshovos, "Demystifying GPU microarchitecture through microbenchmarking," *2010 IEEE International Symposium on Performance Analysis of Systems & Software (ISPASS)*, White Plains, NY, pp. 235-246. 2010. doi: 10.1109/ISPASS.2010.5452013.
[17] A. Jog *et al.*, "OWL: Cooperative Thread Array Aware Scheduling Techniques for Improving GPGPU Performance," in *International Conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS*, vol. 41, pp. 395-406, 2013. doi: 10.1145/2490301.2451158.
[18] NVIDIA, "NVIDIA Tesla P100 The Most Advanced Datacenter GPU," *Exalit Profesional Graphics and HPC*, 2016.
[19] S. Che et al, "Rodinia: A benchmark suite for heterogeneous computing," *2009 IEEE International Symposium on Workload Characterization (IISWC)*, Austin, TX, 2009, pp. 44-54. doi: 10.1109/IISWC.2009.5306797.
[20] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong and T. M. Aamodt, "Analyzing CUDA workloads using a detailed GPU simulator," *2009 IEEE International Symposium on Performance Analysis of Systems and Software*, Boston, MA, pp. 163-174, 2009. doi: 10.1109/ISPASS.2009.4919648.
[21] M. Khairy, Z. Shen, T. M. Aamodt and T. G. Rogers, "Accel-Sim: An Extensible Simulation Framework for Validated GPU Modeling," *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, Valencia, Spain, pp. 473-486. 2020. doi: 10.1109/ISCA45697.2020.00047.
[22] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, and K. Skadron, "A performance study of general-purpose applications on graphics processors using CUDA," *Journal of Parallel and Distributed Computing*, vol. 68, no. 10, pp. 1370-1380, 2008. doi: 10.1016/j.jpdc.2008.05.014.

[23] S. Xiao and W. Feng, "Inter-block GPU communication via fast barrier synchronization," *2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS), Atlanta, GA*, pp. 1-12, 2010. doi: 10.1109/IPDPS.2010.5470477.
[24] W. Feng and S. Xiao, "To GPU synchronize or not GPU synchronize?," *2010 IEEE International Symposium on Circuits and Systems (ISCAS)*, Paris, pp. 3801-3804. 2010. doi: 10.1109/ISCAS.2010.5537722.
[25] S. Che, J. W. Sheaffer, M. Boyer, L. G. Szafaryn, Liang Wang and K. Skadron, "A characterization of the Rodinia benchmark suite with comparison to contemporary CMP workloads," *IEEE International Symposium on Workload Characterization (IISWC'10)*, Atlanta, GA, pp. 1-11, 2010. doi: 10.1109/IISWC.2010.5650274.

BIOGRAPHIES OF AUTHORS

Vo Viet Tan received the B.S. degree from Electronics and Telecomunication Engineering of Ho Chi Minh City University of Technology, Ho Chi Minh, Vietnam in 2018. He is pursuing his M.S. in Electronics and Computer Engineering at Chonnam National University. His research interests include computer architecture, parallel processing, microprocessors, and GPGPUs.

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. He also worked as a Professor at Chonnam National University, Korea from 2007 to 2020. Now he is working as a Professor at School of Computer Sciece and Engineering, Soongsil University, Korea. His research interests include computer systems, embedded systems, mobile systems, computer architecture, SoC design, low power systems, and multiprocessors.