Cache efficiency based dynamic bypassing technique for improving GPU performance

Min Goo Moon[1], **Cheol Hong Kim***[1]
[1]School of Electronics and Computer Engineering,
Chonnam National University,
Gwangju, 61186, Korea
airnia41@gmail.com, chkim22@jnu.ac.kr*

Abstract - *GPUs utilize parallel HW resources to provide good performance for graphics as well as general purpose programs. There have been many researches to improve performance by using GPUs. However, due to the problems such as cache contention, it is not always possible to fully utilize hardware resources in GPUs. In this paper, we propose a simple cache bypassing technique to enhance the resource utilization in GPUs. The proposed bypassing technique that selectively caches memory requests can mitigate these problems. According to our simulation results, the proposed technique reduces the number of accesses to the L1 data cache, which improves performance and reduces power consumption. Compared to baseline GPU architecture, the proposed scheme can increase the average performance by about 100%.*

Keywords: GPU, GPGPU, Performance, Cache bypassing

1 Introduction

A graphics processing unit (GPU) is a processor with powerful hardware resources to perform graphics operations that require high processing power. To achieve high performance, GPUs execute multiple threads in parallel. This is called as thread level parallelism (TLP). TLP allows to efficiently utilize the resources of the GPU. Because the GPU greatly outperforms the CPU for tasks that require large amounts of parallel computation, it can greatly improve overall computing power by performing dedicated graphic operations on the CPU's workload [1,2,3]. As a result, the CPU and the GPU are now used together to improve the performance of recent computing systems.

Recent GPUs can handle not only graphics operations but also general-purpose work. The technique to utilize GPUs for general-purpose applications is called as General Purpose Computation on the Graphics Processing Unit (GPGPU) [4]. GPGPU can utilize the powerful hardware of the GPU to effectively perform general-purpose operations. Therefore, the performance of general-purpose programs can be greatly improved by using GPUs. GPU vendors provide many application programming interfaces (APIs) such as OpenCL [5] and CUDA [6] to facilitate the parallelism for GPUs. These APIs allow programmers to use the GPU more efficiently.

To achieve high performance, GPU programmers tend to enhance thread level parallelism. However, massive thread level parallelism cannot be exploited well for various cases [7]. Because the GPU runs many threads in parallel, a large number of memory requests access the cache in a short period. By contrast, the size of the cache is very small. This causes resource bottleneck problems such as cache contention, resulting in performance degradation due to the stall. These problems decrease the resource utilization of GPUs.

In this paper, we propose a simple cache bypassing technique to overcome these problems. The rest of this paper is organized as follows. Section 2 describes the background on GPU architecture and cache bypassing techniques. Section 3 describes the motivation of the proposed technique and the proposed cache bypassing technique. Section 4 presents our experimental methodology and discusses the experimental results. Finally, we conclude this paper in Section 5.

2 Background

2.1 GPU Architecture

In this work, we use GPU architecture similar to NVIDIA's Fermi GPU architecture. In order to avoid confusion due to the use of different terminologies by various GPU vendors [9,10], we use the terminology for the GPU architecture defined by NVIDIA [8]. GPUs vary in terminology and method of operation depending on the vendors, but the general structure is similar. In conventional GPUs, the performance is improved by performing dedicated operations related to graphics in the workload executed by the CPU. However, modern GPUs allow not only graphics operations but also general-purpose operations.

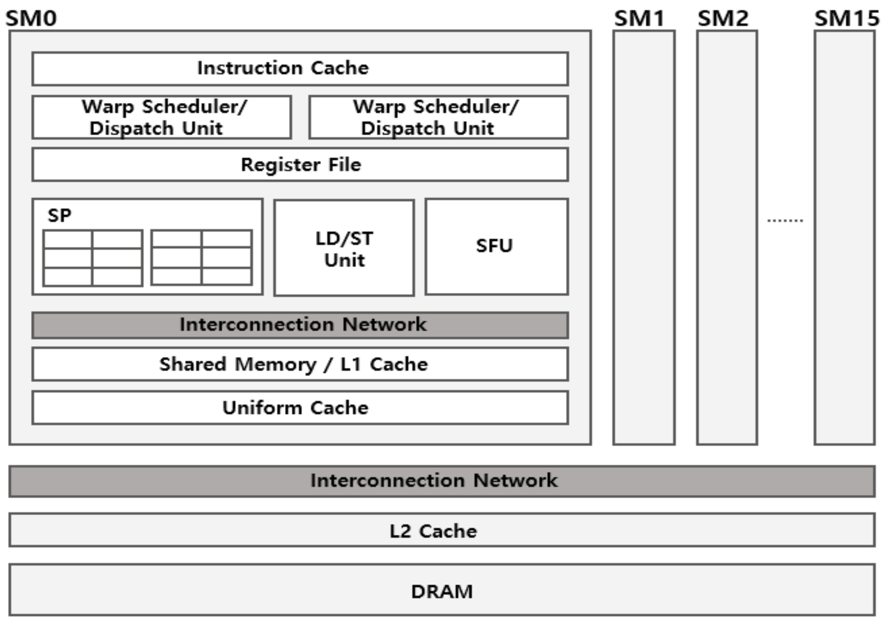

Figure 1. Fermi GPU Architecture

Typical GPGPUs use the single instruction multiple threads (SIMT) execution method and allow multiple operations to be performed simultaneously on a single instruction. The basic unit used in the operation is a thread, and 32 threads can be executed at the same time in general. This set of threads that can be executed at the same time is called as a warp. A set of warps are combined into a cooperative thread arrays (CTAs), and several CTAs form a kernel. One typical GPGPU application consists of one or more kernels.

Figure 1 illustrates Nvidia's Fermi GPU architecture [8]. As shown in the figure, the GPU consist of many SMs (16 for Fermi) and each SM is connected through interconnection network. One SM consists of 32 SPs and special function unit, load store unit, register file, shared memory, and L1 cache. It also includes a warp scheduler and a dispatch unit for allocating the warps to SPs. The SP, called as CUDA core, performs arithmetic operations through an arithmetic unit (ALU) and a floating-point unit (FPU). In the case of complex arithmetic operations such as trigonometric function and square root, it is performed in SFU.

There are L1 data cache, constant cache, texture cache and shared memory in each SM, and L2 cache and DRAM exist outside SM [8,11,12]. The L1 data cache stores data required by the memory requests, thereby reducing the frequency of access to the DRAM requiring long latency. Shared memory has the role of sharing data between threads executed in each SP and facilitates reuse of on-chip data, thereby reducing the traffic to the DRAM. The L2 cache handles all load, store, and texture requests and enables efficient and fast data sharing across the GPU. GPUs typically require a large amount of memory bandwidth because they perform a large amount of parallel operations. Therefore, this kind of memory is organized as hierarchical memory to efficiently process large amounts of data.

2.2 Cache bypassing

Recently, the number of cores in the processor has been increased to improve the performance of computing systems. As the number of cores increases, the cache size also increases because of the increased demand for caches. However, due to limited hardware resources, the cache cannot be infinitely extended. Especially, in GPU architecture, since the large amount of operations are performed in parallel, the size of the L1 data cache is relatively small and the cache utilization rate is low. Cache bypassing technique is the way that the requests are sent to the next memory layer without accessing the cache. In this case, the cache bypassing technique is an effective way to increase the cache efficiency without increasing hardware complexity.

3 Proposed Technique

3.1 Motivation

The GPU has a number of SPs (512 in the case of the Fermi) to allow simultaneous execution of multiple threads for parallel operations, and adopts a hierarchical memory structure to effectively utilize the large memory bandwidth. This architecture allows multiple threads to run at the same time, and the cache has a broad line to maximize throughput for parallel operations [13]. Maximizing parallelism can improve the performance of the GPU, but it can be a burden on the memory system. A single warp contains 32 threads, each thread generates memory requests associated with the operation. If several memory requests are targeted to the same cache line, the requests can be merged [14]. However, if memory requests are designated to different cache lines, up to 32 memory requests can occur for a single warp. In this case, the cache contention increases. Increased cache contention causes cache lines to be replaced, evicting the data that can be reused. This increases the number of accesses to lower level memory, resulting in increased delay because the next request will be stalled until previous requests occupying the cache are completed. This problem is called as memory divergence [15]. In GPGPUs, since many threads are executed in parallel, cache contention problem can be serious. As a result, the number of misses in the L1 cache increases, so additional accesses to lower level memory are required and the latency for executing warp is greatly increased. Additional power consumption causes the increase of total power consumption. Therefore, there is a need for methods that can maximize the utilization of GPU resources while minimizing power consumption in consideration of cache contention problems. Cache bypassing technique is one of the simplest techniques for this purpose.

3.2 Proposed Architecture

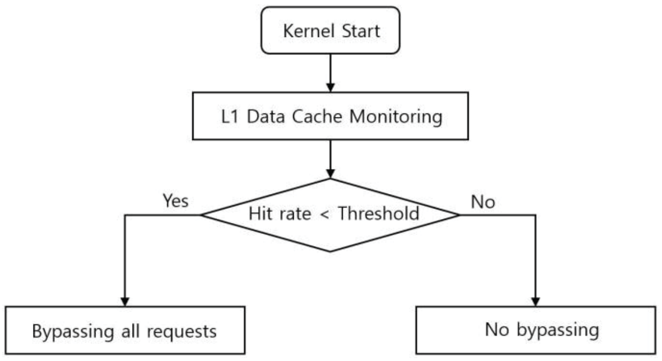

Figure 2. Proposed bypassing scheme

The proposed technique utilizes access information to reduce unnecessary accesses. When the GPU kernel program is executed, it measures the information about the number of accesses and hit/miss rates to the L1 data cache during the monitoring period. In this work, 3,000 cycles are used as the monitoring period. At the end of the monitoring period, the hit rate is calculated by using the measured information and it is compared with the threshold. If the hit rate for the L1 data cache is less than the threshold, all requests are bypassed during the remaining kernel execution time. If the hit rate is low, it causes problems such as stalling, so bypassing is effective for improving the performance. If the hit rate is bigger than the threshold value, no bypassing is required.

To achieve optimal performance, the proposed method dynamically determines whether to bypass or access the cache. The proposed method requires additional hardware area to store measured access and hit/miss information for the L1 data cache during the monitoring period and threshold value. Therefore, the hardware overhead is very small.

4 Experiments

4.1 Experimental Methodology

In this section, we briefly describe the experimental methodology. We implemented the proposed bypassing method through GPGPU-SIM [16]. GPGPU-SIM is a proven simulator that is widely used in the GPU architecture research community. The baseline architecture is modeled based on NVIDIA's Fermi GPU. The simulated configuration is shown in Table 1. In this work, we use four kinds of benchmark programs to analyze the performance of various programs. The benchmark programs used in this work are ISPASS [16], RODINIA [17], NVIDIA SDK [18], PolyBench [19], PARBOIL [20]. The selected programs are lud, hotspot, reduction, needle, bfs, mergesort, atax, bicg, simplemultigpu, sgemm. In this paper, the names of each benchmark are abbreviated as LUD, HS, RD, NW, BFS, MS, ATAX, BICG, SMG, SGM.

Table 1. Baseline GPGPU-Sim configuration

Parameter	Value
Number of SM	15
Warp Size	32
Number of threads/SM	1024
L1 Data cache	16KB, 4-way, 128byte lines
L2 Data cache	16KB, 8-way, 128byte lines
Shared Memory	16KB

Table 2. Benchmarks

Benchmark	Application
ISPASS [16]	bfs
Rodinia [17]	lud
Rodinia [17]	hotspot
Rodinia [17]	needle
Nvidia SDK [18]	reduction
Nvidia SDK [18]	mergesort
Nvidia SDK [18]	simplemultigpu
PolyBench [19]	atax
PolyBench [19]	bicg
Parboil [20]	segmm

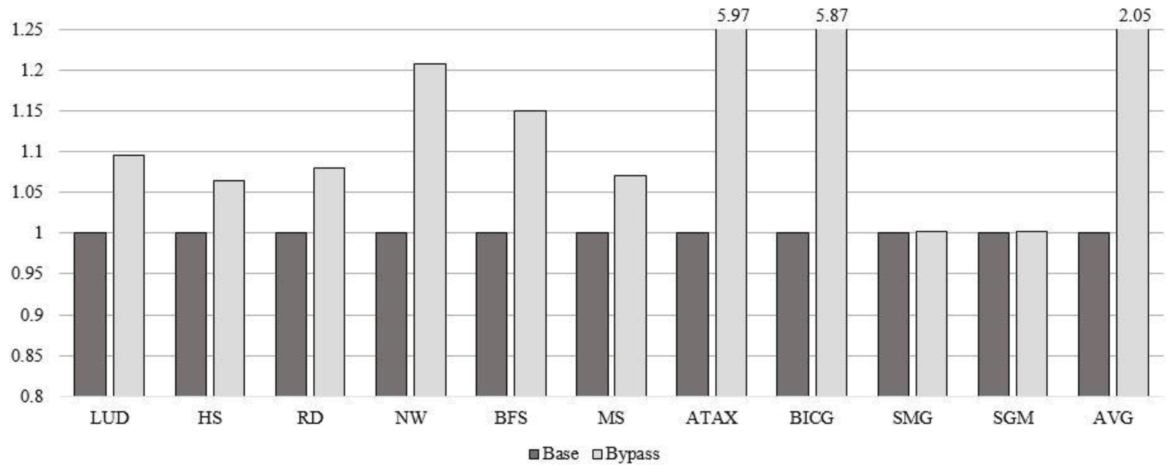

Figure 3. IPC

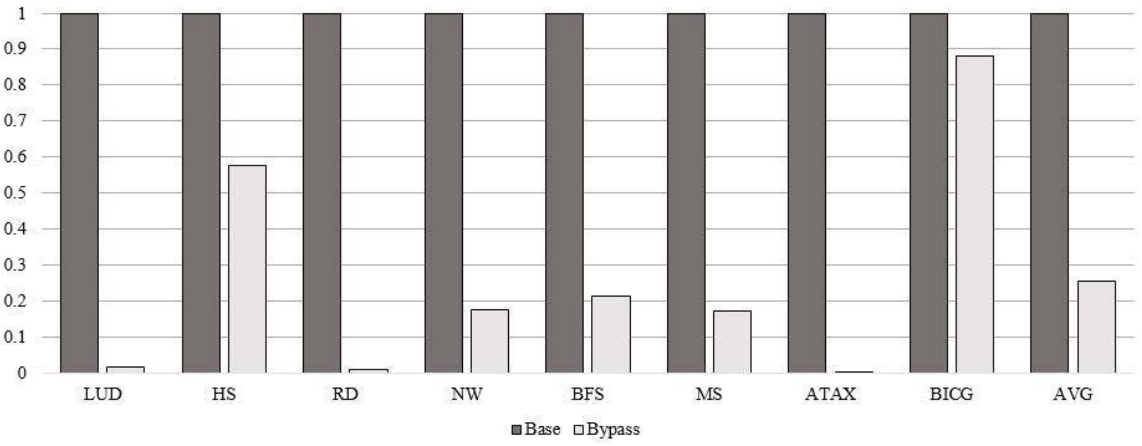

Figure 4. Power efficiency

4.2 Evaluation Results

In this section, we analyze the simulation results for the proposed bypassing technique. Our experiments are performed on various benchmark programs for NVIDIA Fermi GPU modeled using GPGPU-SIM. In this paper, the performance is normalized to the baseline GPU architecture.

Figure 3 shows the performance comparison when the proposed bypassing technique is applied. The vertical axis in the graph represents the normalized performance (IPC) and the horizontal axis represents the benchmark used in the experiments. As shown in the graph, for most of benchmarks, the proposed bypassing technique provides improved performance compared to the baseline architecture. Benchmark applications with enhanced performance includes a kernel with very low hit rate of less than 10% for the L1 data cache. When the cache utilization is poor, applying the proposed bypassing technique reduces the delay due to cache contention and resource shortage such as MSHR, which shortens the latency and improves the overall performance. In particular, the proposed technique shows great performance improvement for ATAX and BICG. This is because the applications have a lot of pipeline stalls due to memory divergence. ATAX and BICG have a total stall of about 1.4 billion. This is about 10 times more than the total number of execution instructions of the application, which means that the performance degradation due to the stall is huge. Therefore, when the proposed bypassing technique is applied, the stalls are reduced and the performance is greatly enhanced. When the proposed bypassing technique is applied, the stall is reduced by about 80%. However, the proposed bypassing technique shows little performance change for SMG and SGM applications, which are cache insensitive programs. Cache insensitive applications are less affected by bypassing because they have less memory request instructions than computing instructions. Except SMG and SGM, the performance is improved by at least 6.55% (HS) and up to 497.51% (ATAX) by adopting our proposed technique. On average, the proposed bypassing technique can improve the performance by about 100%.

Figure 4 shows the change in the power consumption of the L1 data cache, which is reduced by applying the proposed bypassing technique. The vertical axis in the graph represents the consumed power and the horizontal axis represents the benchmark used in the experiment. The power consumption is normalized to the baseline GPU architecture. As shown in the graph, the proposed bypassing technique reduces the power consumed in the L1 data cache. If the kernel running in the program has both high hit rate kernel and low hit rate kernel, the total power consumption can be decreased as shown in the graph for HS, NW, BFS, MS and BICG. If the hit rate of all kernels is low, there is little power consumption because the L1 data cache is accessed only during the monitoring period and then bypassed. On average, only about 20% of total power is consumed in the L1 data cache when the proposed bypassing technique is applied. The power consumption of L1 data cache accounts for about 5% of the total power consumed in the baseline architecture.

5 Conclusions

Modern GPUs are designed for high-speed parallel execution by utilizing huge computing resources. However, to take full advantage of GPU resources, we should consider cache contention and pipeline stall issues caused by small data caches. In this paper, we proposed a cache bypassing technique considering these problems. The purpose of the proposed bypassing scheme is to reduce additional cache accesses and power consumption due to the small L1 data cache size. The proposed technique monitors the L1 data cache during sampling period for each kernel and enables bypassing the cache when the hit rate is low. Experimental results showed that the performance can be improved by about 100% for the applications with low hit rate for L1 data cache. As the number of accesses to the L1 data cache decreases, the additional power consumption at the corresponding part also decreases. The proposed technique required only small area to store the hit count and the threshold value to be measured during execution, so the hardware overhead is very small.

Acknowledgements

This research was supported by the MSIT(Ministry of Science and ICT), Korea, under the ITRC(Information Technology Research Center) support program(IITP-2018-2016-0-00314) supervised by the IITP(Institute for Information & communications Technology Promotion), and it was also supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2018R1A2B6005740).

6 Reference

[1] S. Ryoo, C. Rodrigues, S. Baghsorkhi, S. Stone, D. Kirk and W. Hwu, “Optimization Principles and Application Performance Evaluation of a Multithreaded GPU using CUDA”, In the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP), pp. 73-82, 2008.

[2] Myung Kuk Yoon, Keunsoo Kim, Sangpil Lee, Won Woo Ro and Murali Annavaram, “Virtual Thread: Maximizing Thread-Level Parallelism beyond GPU Scheduling Limit”, ACM/IEEE 43rd Annual International Symposium on Computer Architecture, 2016.

[3] V. W. Lee, C. K. Kim, J. Chhugani, M. Deisher, D. H. Kim, A. D. Nguyen, N. Satish, M. Smelyanskiy, S. Chennupaty, P. Hammarlund, R. Singhal and P. Dubey, “Debunking the 100X GPU vs. CPU Myth: An Evaluation of Throughput Computing on CPU and GPU”, International Symposium on Computer Architecture, pp. 451-460, 2010.

[4] General-purpose computation on graphics hardware, http://www.gpgpu.org

[5] OpenCL, http://www.khronos.org/opencl/

[6] NVIDIA CUDA Programming, http://www.nvidia.com/object/cuda_home/new.html

[7] A. Jog, E. Bolotin, Z. Guz, M. Parker, S. W. Keckler, M. T. Kandemir, C. R. Das, “Application-aware Memory System for Fair and Efficient Execution of Concurrent GPGPU Application”, 7th Workshop on General Purpose Processing Using GPUs, March 2014.

[8] NVIDIA’s Next Generation CUDA Compute Architecture: Fermi, www.nvidia.com/content/pdf/fermi_white_papers/nvidia_fermi_compute_ architecture_whitepaper.pdf

[9] NVIDA Co. Ltd., http://www.nvidia.com/

[10] AMD (Advanced Micro Devices) Inc., http://www.amd.com/

[11] H. J. Choi, C. H. Kim, “Performance Evaluation of the GPU Architecture Executing Parallel Applications,” Journal of the Korea Contents Association, Vol.12, No.5, pp.10-21, 2012

[12] H. J. Choi, G. M. Kim, C. H. Kim, “Analysis on the GPU Performance according to Hierarchical Memory Organization” Journal of the Korea Contents Association, Vol.14, No.3, pp.22-32, 2014

[13] Kayvon Fatahalian, Mike Houston, “A Closer Look at GPUs”, Communication of the ACM 51, pp. 50–57, Oct 2008

[14] N. B. Lakshminarayana, H. S. Kim, “Effect of Instruction Fetch and Memory Scheduling on GPU Performance”, Workshop on Language, Compiler, and Architecture Support for GPGPU (in conjunction with HPCA/PPoPP 2010), 2010.

[15] Bin Wang, “Mitigating GPU Memory Divergence for Data-Intensive Applications”, A dissertation submitted to the Graduate Faculty of Auburn University in partial fulfillment of the requirements for the Degree of Doctor of Philosophy, 2015.

[16] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt, “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.

[17] S. Che, M. Boyer, J. Meng, D. Tarjan, J. Sheaffer, S.-H. Lee, K. Skadron, “Rodinia: a benchmark suite for heterogeneous computing”, The International Symposium on Workload Characterization (IISWC), pp.44–54, 2009.

[18] NVIDA, CUDA SDK, http://developer.nvidia.com/gpu-computing-sdk.

[29] S. Grauer-Gray, L. Xu, R. Searles, S. Ayalasomayajula, and J. Cavazos, “Auto-tuning a High-Level Language Targeted to GPU Codes” Innovative Parallel Computing (InPar), 2012.

[20] J. A. Stratton, C. Rodrigues, I.-J. Sung, N. Obeid, L.-W. Chang, N. Anssari, G. D. Liu, and W.-M. W. Hwu, The IMPACT Research Group, “Parboil Benchmark Suite.” [Online]. http://impact.crhc.illinois.edu/Parboil/parboil.aspx