Journal of The Korea Society of Computer and Information
Vol. 23 No. 10, pp. 1-9, October 2018
www.ksci.re.kr
https://doi.org/10.9708/jksci.2018.23.10.001

Workload Characteristics-based L1 Data Cache Switching-off Mechanism for GPUs

Thuan Cong Do*, Gwang Bok Kim**, Cheol Hong Kim**

Abstract

Modern graphics processing units (GPUs) have become one of the most attractive platforms in exploiting high thread level parallelism with the support of new programming tools such as CUDA and OpenCL. Recent GPUs has applied cache hierarchy to support irregular memory access patterns; however, L1 data cache (L1D) exhibits poor efficiency in the GPU. This paper shows that the L1D does not always positively affect the applications in terms of performance and energy efficiency for the GPU. The performance of the GPU is even harmed by using the L1D for lots of applications. Our proposed technique exploits the characteristics of the currently-executed applications to predict the performance impact of the L1D on the GPU and then decides whether to continuously use the cache for the application or not. Our experimental results show that the proposed technique improves the GPU performance by 9.4% and saves up to 52.1% of the power consumption in the L1D.

▸Keyword: GPU, Cache Bypass, High Performance, Energy Efficiency, L1 Data Cache

I. Introduction

Modern Graphics Processing Units (GPUs) are employed for handling not only graphics but also general-purpose computations to utilize huge hardware resources, known as General Purpose Computing on Graphics Processing Units (GPGPUs). Cache memories have been employed in recent GPUs to support diverse access patterns, which are frequently exhibited in general-purpose applications. In general, caches can deliver performance improvement to the GPU for a majority of applications; however, GPU caches have to confront with many challenges caused by the limitations of memory subsystem as well as cache hardware budget. In addition, the design of GPU caches is still based on the latency-optimized CPU caches [1] while the number of memory requests issued by GPGPU applications is much larger than the number issued by CPU applications. Consequently, the low per-thread capacity and low associativity of GPU caches can cause a system bottleneck and finally performance degradation [2, 3].

In this work, we show that the L1D can harm the performance of many other applications, even worse than a no-L1D GPU. Furthermore, inappropriate usage of the L1D consumes unnecessary energy for several applications. Based on these observations, this paper proposes a new mechanism (called AdmL1D) that is able to switch off the L1D in the GPU depending on the characteristics of the currently-executed applications. The proposed AdmL1D tracks several parameters representing the characteristics of the currently-executed application during its execution time, and then exploits these parameters as a hint to predict the impact of the L1D on the GPU performance. The proposed mechanism

• First Author: Thuan Cong Do, Corresponding Author: Cheol Hong Kim
*Do Cong Thuan (congthuan@korea.ac.kr), Dept. of Computer Science and Engineering, Korea University
**Gwang Bok Kim (loopaz63@gmail.com), School of Electronics and Computer Engineering, Chonnam National University
**Cheol Hong Kim (chkim22@chonnam.ac.kr), School of Electronics and Computer Engineering, Chonnam National University
• Received: 2018. 07. 25, Revised: 2018. 10. 05, Accepted: 2018. 10. 17.
• This study was financially supported by Chonnam National University (Grant number: 2017-2727).

makes the decision about either continuously using or switching off the cache. Our experiments show that the proposed AdmL1D technique can improve the overall GPU performance and significantly reduce the L1D energy consumption for applications those cannot benefit from the L1D. The simulation results show that the AdmL1D provides a speedup of 9.4% over the baseline and saves 52.1% L1D power consumption on average. Moreover, the AdmL1D does not degrade the performance for all simulated applications.

The rest of this paper is organized as follows. Section 2 discusses related work. We briefly describe the GPU architecture and our motivation in Section 3. Section 4 presents the proposed L1 data cache bypassing technique for GPUs. Section 5 describes simulation methodology and detailed experimental results. Finally, Section 6 concludes this paper.

II. Related Work

1. Related works

Recent GPUs have been widely adopted to handle general-purpose applications as well as graphics applications. A large number of works have been proposed with the aim at fully utilizing the potentials of GPU hardware resources while a fewer works have focused on the energy reduction of the GPU. Warp/CTA scheduling has gained the most attention due to their vital role in improving the GPU performance. Various scheduling schemes were proposed to increase the utilization of GPU hardware resources [4, 5, 6, 7, 8]. Control-flow divergence is also one of the main problems in up-to-date GPU architectures. Compared to other techniques, Fung et al. [9] proposed one of the most effective techniques for control-flow divergence problem, which combine the threads from different warps to address the hardware underutilization caused by branch divergence.

The hardware budget for GPU caches limits the effectiveness of cache memory in handling irregular memory accesses. For this reason, proposing efficient cache management techniques for the GPU is still challenging. Recently, several works return to a traditional cache bypassing solution for selectively bypassing memory requests. Cache bypassing has been widely applied to CPUs and is recognized as an effective method to mitigate cache contention and resource congestion. In general, cache-bypassing techniques can be classified into two types of approaches including static [15] and dynamic [16, 17, 19, 21]. Jia et al. [1] proposed a hardware structure called memory request prioritization buffer (MRPB), which employs request reordering and cache bypassing, to avoid a system bottleneck in GPU caches. Meanwhile, Chen et al. [23] and Duong et al. [24] proposed an adaptive cache management technique by combining the protection distance (PD) with cache bypassing to improve the cache performance. Xie et al. [25] proposed a coordinated static and dynamic cache bypassing technique to identify the global loads that indicate strong preferences for caching or bypassing through profiling.

III. Background

3.1 Baseline GPU Architecture

This section briefly describes the baseline GPU architecture, which is based on a typical NVIDIA GPU. Further details on these architectures can be found in previous works [9, 13]. The GPU consists of many streaming multiprocessors (SMs); each typically has SIMT (Single Instruction Multiple Threads) lanes of 32. Inside the GPU, an interconnection network is used to connect SMs together. The powerful computation ability of the GPU partly comes from a huge register file supporting parallel computation. With the on-chip register file, each SM can accommodate a large number of threads. Similar to CPUs, the GPU also have several special function units (SFUs) and Load/Store units (L/SUs), which are responsible for executing instructions.

The CUDA platform is used to parallelize the applications. A CUDA application typically consists of many kernels, each kernel includes groups of threads known as CTAs. Within a CTA, threads are grouped into warps of 32 threads, to be executed in lockstep fashion with one instruction at one time. A CTA is an abstraction that is required to encapsulate all synchronization and barrier primitives among warps [20], helping CTAs to be executed on SM in any order. Scheduling in a GPU is performed as three-step process. In the first step, a kernel is launched on the GPU. Next, the global CTA

www.kci.go.kr

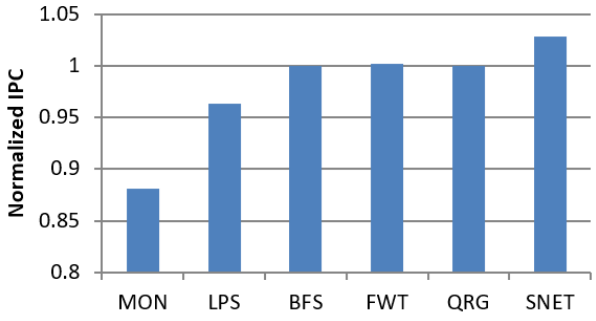
scheduler (e.g. GigaThread [14]) is responsible for assigning CTAs to all available SMs in the second step. CTAs are assigned according to a simple round-robin fashion. Hardware resources limit the maximum number of CTAs assigned per SM [18]. After the assignment of CTAs, in the third step, warps associated with the launched CTAs are scheduled to SIMT lanes of the corresponding SM.

The baseline GPU has multiple levels of memories, from L1 memories to DRAM. Each SM contains various types of on-chip caches, including special purpose caches (e.g. read-only, texture, constant and data). The L1D is private per-SM and used to support irregular data accesses. All threads of sibling warps can share L1 memories. Meanwhile, all of the CTAs in a kernel can access the shared L2 cache and off-chip DRAM. Each memory controller is associated with a slice of shared L2 cache bank. Memory controllers are responsible for scheduling memory requests caused by L2 cache miss to the DRAM. In the baseline GPU, the L1D works as the central point of coherency since the SIMD cores are connected to interconnection network through L1Ds. In both cache levels, missed requests are recorded by the Miss Status Holding Registers (MSHRs) and then sent to the next level of the memory hierarchy [18].

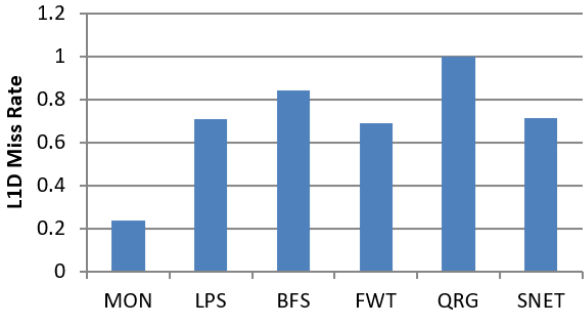
3.2 Motivation

In general, many previous works [16, 21] supposed that general-purpose applications take advantage from cache memories, especially from the L1D in terms of performance. However, we cannot be sure that this point of view is always correct in GPUs. We suppose that the GPU caches usually have unpredictable impact on the performance. In this work, we only focus on the performance impact of the L1D on general-purpose applications except graphics applications. We analyze the performance of several applications when they are executed on the GPU with and without the L1D. We use GPGPU-sim [18] to model NVIDIA-like GPU and benchmarks from CUDA SDK [22], Rodinia [26] and ISPASS [18] in this work.

Fig. 1(a) shows the performance of general-purpose applications executed on the GPU without L1D normalized to the baseline. Note that the GPU without L1D means that all memory requests to the L1D will be directly forwarded to next level of memory hierarchy (e.g. L2 cache). Fig. 1(b) presents the miss rates of the L1D when applications are executed on the GPU with L1D. From these figures, although the L1D has positive impact in the GPU memory architecture for some of applications, it can even harm the performance of some other applications. For example, in the case of FWT, it cannot gain any performance advantage from the L1D despite the fact that more than 30% of memory requests hit the L1D. However, with a similar hit rate, SNET even benefits from bypassing the L1D. On the other hand, the performance of LPS is reduced when GPU does not employ the L1D, as we expected. Based on these results, we classify the applications into two types depending on the advantage they get from the L1D. Type-P denotes the applications that the L1D has positive performance impact while Type-N represents the applications that the L1D has non-positive performance impact. According to these results, we can know that the impact of L1D on the performance of general-purpose applications is unpredictable.



(a) Normalized IPC



(b) L1D miss rate

Fig. 1. Normalized IPC and L1D miss rates for GPGPU workloads

IV. Proposed Technique

The number of times memory requests cannot be

www.kci.go.kr

accepted by the L1D (req-fail-num) during execution time called warm-up-period is monitored. We find that the ratio of the req-fail-num to the number of the L1D cache accesses (L1D-acc-num), called R-to-A ratio, can represent the cache efficiency well at run time. A memory request is not handled due to the unavailability of necessary resources, for example, the cache space is all reserved or the miss queue is full. As the hardware budget for the L1D is too small compared to the number of memory requests from many SMs of the GPU, a large number of memory requests cannot be processed and have to retried in next cycles. When the number of memory requests not handled is greater than the threshold, it causes pipeline stalls and degrades the performance. Therefore, the req-fail-num and L1D-acc-num parameters are tracked and used to evaluate the efficiency of the L1D for the AdmL1D.

Fig. 2 represents the R-to-A ratio of various applications over one million cycles. We varied the sampling period length depending on the length of the execution time of applications. From this graph, we can observe that there is a wide gap between the R-to-A ratio of Type-P applications and Type-N applications. Type-P applications (e.g. MON, LPS and QRG) can benefit from the L1D, therefore, they have very low R-to-A ratios, around zero. Meanwhile, Type-N applications (e.g. FWT, SNET and BFS) those cannot benefit from the L1D, have high R-to-A ratios. Furthermore, Fig. 2 shows that the R-to-A ratio of all applications is stable at the beginning of execution for a long period of time. Based on our experiments, we empirically set cut-off of the R-to-A ratio for 3, called B-threshold. This means that an application with the R-to-A ratio greater than 3 is considered as Type-N applications. When the AdmL1D is applied, the req-fail-num and L1D-acc-num parameters are tracked during the warm-up-period.

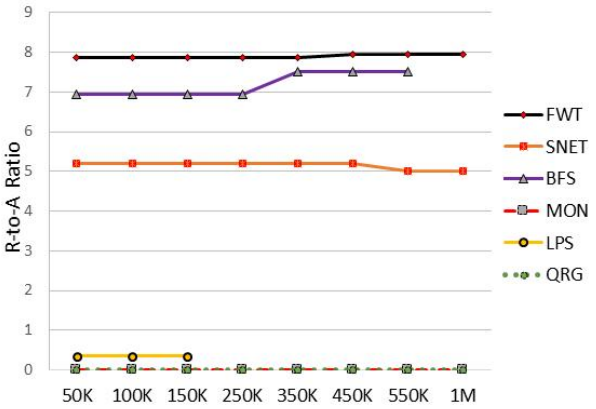


Fig. 2. R-to-A ratio

To prevent the performance degradation due to the L1D, we propose a method that is able to identify what type of applications cannot benefit from the L1D and what type of applications can benefit from the L1D. When the identification step finishes, the proposed AdmL1D decides to continue employing or bypassing the L1D. Note that the identification process needs to provide an accurate result in a short period to maximize the potential performance gains of the proposed technique. The right side of Fig. 3 illustrates the detailed mechanism with the

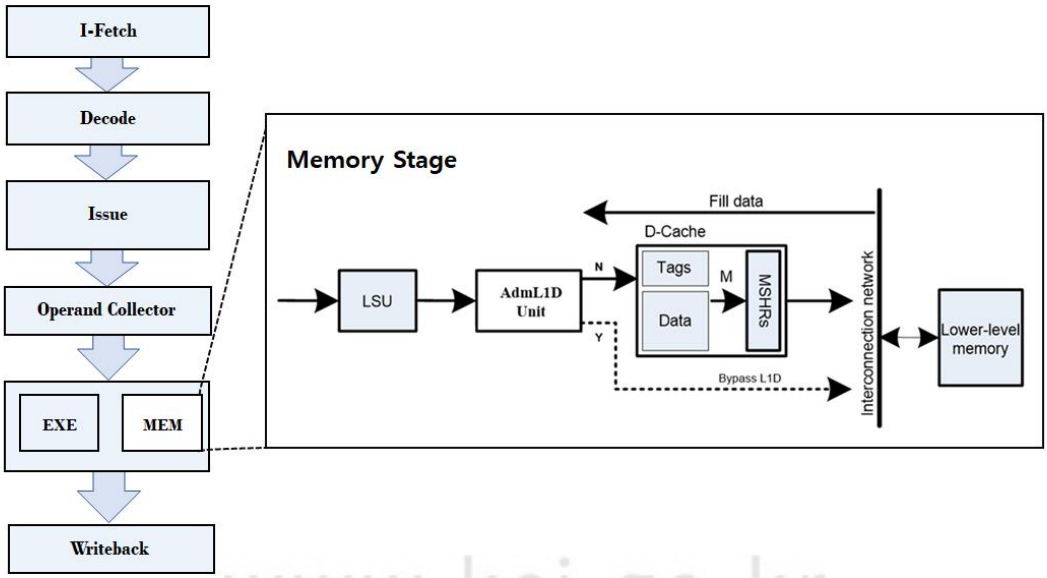


Fig. 3. Pipeline architecture at MEM stage with proposed AdmL1D

www.kci.go.kr

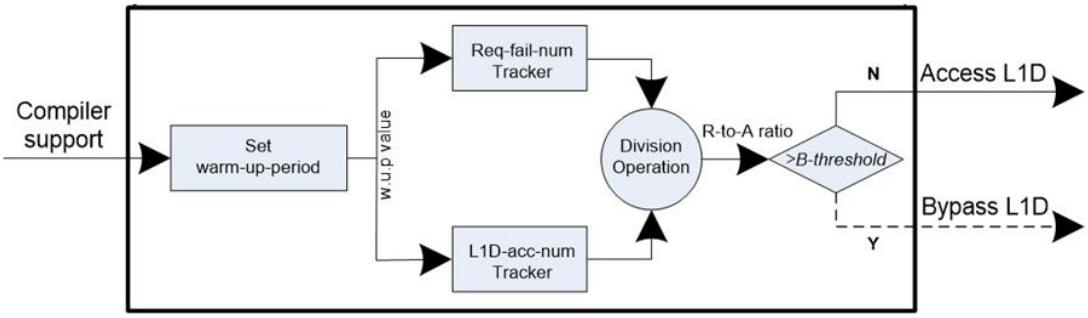


Fig. 4. Diagram of AdmL1D

AdmL1D at the MEM stage. Many complicated cache management techniques [10, 11, 17] cannot demonstrate their effectiveness due to the limited hardware budget of L1D. We suppose that all bypassing the L1D when GPU is executing applications on which the L1D has negative performance impact could provide two potential advantages including performance improvement and energy savings. The challenge of the proposed technique is to recognize the negative impact of the L1D on the application's performance at runt time. As shown in the Fig. 4, the GPU cache management is modified with extra hardware logic, AdmL1D unit. The AdmL1D consists of three major parts: (1) a hardware logic unit that is used for setting up initial parameters, (2) a hardware logic that is used to count the number of request fails (req-fail-num) and L1D accesses (L1D-acc-num) and then make the R-to-A-ratio (the ratio of req-fail-nums to L1D-acc-num), and (3) a comparator that compares the R-to-A-ratio with the B-threshold to make the bypassing decision. The output of AdmL1D unit decides whether the memory requests can access the L1D or not.

The hardware logic unit (1) is responsible for setting the warm-up-period. During this period, the AdmL1D unit collects the information about the L1D cache usage. warm-up-period is an important parameter that can affect the efficiency of the proposed technique. The value of this parameter depends on the execution time of each application and is set based on the experiments with the support of compiler. The execution time of the application can be known exactly after the application is completed. However, with the support of modern compilers, the calculation of execution time can be simply obtained. For instance, compliers can calculate the number of instructions when the application is compiled, thus, the execution time can be estimated quite accurately. Note that the AdmL1D does not need an exact estimation of application's execution time. We can use two counters for the hardware logic unit (2) to track the req-fail-num and L1D-acc-num. The counting process is carried out during the warm-up-period. In fact, there are several parameters instead of cache-hit rates to predict the cache usage, but we see that req-fail-num reflects the most accurate cache usage information in a short period. At the end of warm-up-period, the R-to-A ratio is calculated and then it is compared to the B-threshold to make the decision about bypassing the L1D. The hardware logic unit (3) can be simply implemented by using a comparator. If R-to-A ratio is greater than B-threshold, since then all memory requests will bypass the L1D and be forwarded to the lower level memories, meaning that the L1D is disabled. Otherwise, memory requests access the L1D as normal.

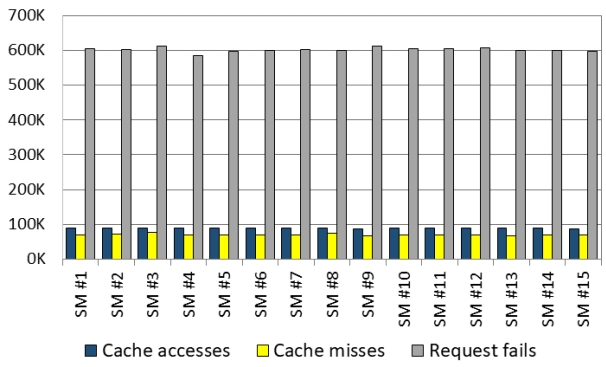


Fig. 5. L1D behavior across SMs of GPU when BFS is executed

Modern GPU architectures contain many SMs, which have the same hardware logic. Therefore, any proposal at SM-level is usually applied for all SMs of the GPU. However, same hardware logic does not guarantee the same cache behavior when applications are executed. In case the L1D cache behavior of each SM is very different, it can affect the correctness of our proposed mechanism. In this work, we experimented on many benchmarks to examine the cache behavior of all SMs.

www.kci.go.kr

We find that the cache behavior among all of the L1D caches of the GPU is almost similar. For example, Fig. 5 shows the number of cache accesses, the number of the cache misses and the number of request fails at the L1D among various SMs when BFS is executed. According to these results, we can know that the L1D caches of GPU exhibit similar cache behaviors. Therefore, the AdmL1D can be applied for all of the SMs of the GPU.

IV. Experiments

We model the GPU with 15 shader cores connected to 6 memory controllers in this work. We use a detailed GPGPU simulator (GPGPU-sim v3.2) [18] in our evaluation. The configuration parameters are described in Table 1. The simulator was modified to implement the AdmL1D. We can see that the parameters for the baseline are based on a generic NVIDIA GPU [2]. However, our technique can be applied for all GPU architectures. We use GPUWattch [28] to evaluate power consumption.

5.1 Simulation Methodology

We consider a wide range of GPGPU workloads, including the applications from CUDA SDK [22], Rodinia [26] and ISPASS [18] as summarized in Table 2. The workloads are selected from simple to complex. Based on the characteristics of applications, the applications are classified into 2 types, Type-P and Type-N. For all benchmarks, the simulation is done until the completion of the workloads.

The AdmL1D has to firstly recognize to which type the currently-executed application is belonged, Type-P or Type-N. We analyze the characteristics of the applications in order to find out the appropriate parameters for the AdmL1D by using 18 GPGPU applications.

Table 1. Configuration of simulated system

Parameter	Value
SIMT Core SIMT Width Resources/Core	15 cores / 32 threads 1024 threads, 48 warps, 32768 registers
Shared memory L1 Data Cache L1 Inst. Cache L2 Cache	48KB, 32 banks 16KB per core, 32-sets/4-ways, 128B line 2KB per core, 4-sets/4-ways 768KB, 128KB/bank, 64-sets/8-ways/6-banks, 128B line
Features	Coalescing enabled, 32 MSHRs/core
Scheduling	LRR warp scheduling, RR CTA scheduling

5.2 Performance Improvement

Fig. 6 presents the impact of the AdmL1D on GPU performance for various GPGPU applications. The IPC is normalized to the baseline GPU architecture. Compared to the baseline, the AdmL1D provides performance improvement by 9.4% on average for Type-N applications. Remarkably, the performance improvement is up to 40% for SPRO and 13% for BFS. The reason behind the high IPC improvement is likely due to the very high R-to-A ratios of these cases. This is reasonable because high R-to-A ratio also represent high degree of negative impact of the L1D on GPU performance. The performance

Table 2. Evaluated GPGPU applications

Type	Application	Abbr	IPC	L1D Pwr	Num Inst
P	DirXTextureCompr[22]	DXTC	675.27	0.11	12365623428
P	BinomialOptions[22]	BIOP	820.25	0.18	8787834880
P	MersenneTwister[22]	MT	320.34	0.95	16657251699
P	MonteCarlo[22]	MON	558.84	1.41	885487448
P	3DLaplaceSolver[22]	LPS	468.98	2.70	62488928
P	StoreGPU[22]	STO	345.36	0.30	115015681
P	MUMmerGPU[22]	MUM	161.31	6.74	21357289
P	EstimatePInlineP[22]	EIP	138.75	0.90	2271007971
P	SingleAsianOptionP[22]	SAOP	162.34	0.12	2807383317
P	QuasiRandGenerator[22]	QRG	572.53	0.41	7859453952
N	FastWalshTransform[22]	FWT	374.29	1.19	3657859072
N	MergeSort[22]	MER	519.81	2.14	12651411314
N	Histogram[22]	HIG	481.49	0.85	27008528592
N	LIBORMonteCarlo[18]	LIB	225.36	0.63	879452160
N	BreadthFirstSearch[26]	BFS	15.85	1.72	12299393

www.kci.go.kr

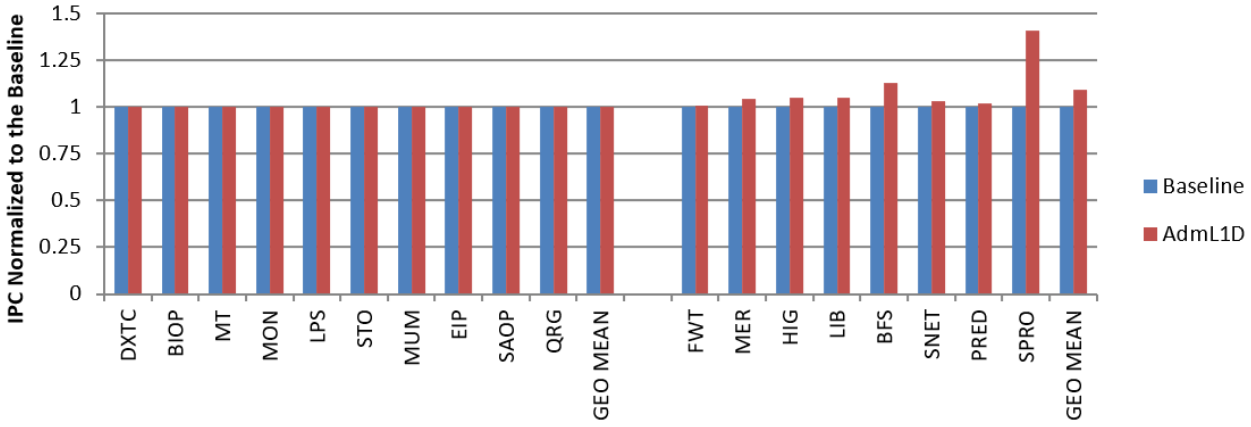


Fig. 6 Performance of AdmL1D for various GPGPU workloads

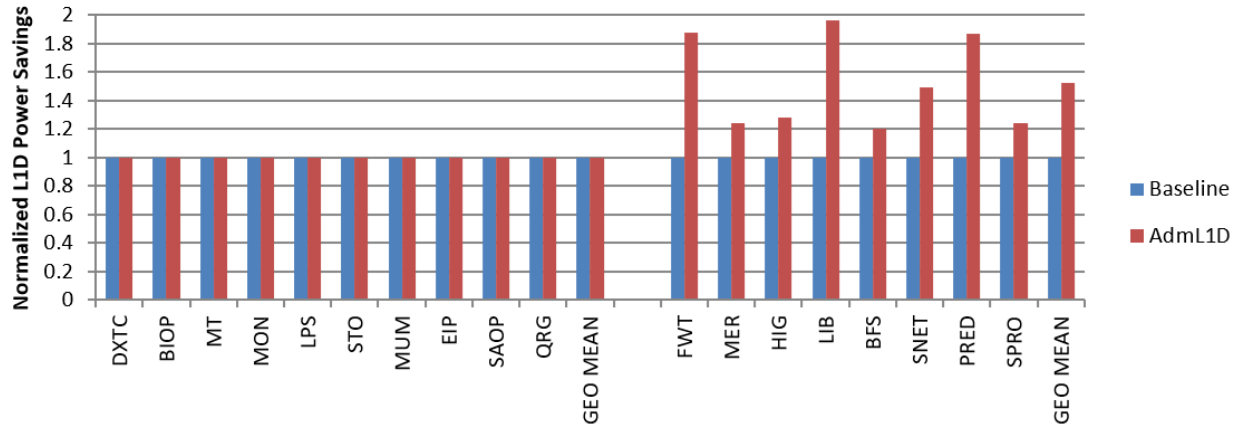


Fig. 7 L1D power savings for various GPGPU workloads

for Type-P applications is maintained that proves the ability of AdmL1D in predicting the performance impact of L1D depending on the characteristics of applications and accurately bypassing the L1D at runtime. Although the L1D can harm the performance of many applications, it is still important to GPUs. According to our experiments, the performance of the GPU is degraded by 8.6% on average when the GPU does not use the L1D for all applications.

5.3 Energy Efficiency

By bypassing the L1D when applications cannot benefit from the L1D, the AdmL1D also provides significant energy reduction of the GPU. Fig. 7 shows the percentage of power savings at the L1D that the AdmL1D can achieve. The results are normalized to the baseline. On average, the AdmL1D can reduce up to 52.1% of the L1D power consumption for Type-N applications. The L1D power consumption in cases of PRED, FWT and LIB can be saved more than 80%. Note that the AdmL1D also provides 9.4% speedup for these applications. This translates into 58.5% reduction of the L1D energy consumption.

The power savings at the L1D contributes to the total power savings of the GPU. We try to put our energy savings in context of high level GPU power model. We assume that a modern GPU chip consumes 150 Watts [27, 28]. If there is a third of this power is spent on leakage [27, 28], dynamic power will be two thirds of the total power, 100 Watts. The memory system usually consumes 30% of dynamic power and the SMs consume 70% of dynamic power [12]. The L1D power is contributed to the SM power because the L1D is private per SM. As our detailed evaluation shows that the AdmL1D saves 52.1% of the L1D power, roughly equivalent to 1% of the total chip power for Type-N applications, this represents 2.1% of SM power and 1.5% of chip-wide dynamic power, or 1.5 Watts. Because that L1 data cache is accessed almost same with baseline architecture for Type-N applications,

By using our proposed architecture, unnecessary accesses which occur dynamic power consumption can be

www.kci.go.kr

reduced. Our bypass technique improves the performance of GPUs by preventing dynamic energy consumption due to unnecessary accesses. Our proposed technique utilizes the characteristics of workload in order to reduce the number of accesses to L1 data cache by predicting data reuse for each cache block. Note that in terms of energy efficiency, our technique can improve further because it also provides the speedup of 9.4%.

V. Conclusions

In this paper, we investigated the impact of L1D on the performance of GPGPU applications. We observed that the L1D does not always have positive effects on the performance for many applications. The L1D even harms the GPU performance for several applications. Therefore, we proposed a mechanism that can bypass the L1D at runtime when the mechanism recognizes that currently-executed application cannot benefit from the L1D. The decision of bypassing the L1D only depends on the characteristics of executed applications. Our proposed technique can provide two main advantages in terms of performance improvement and energy savings without slowing down any applications. Experimental evaluations show that the proposed AdmL1D technique can improve the performance of GPU by 9.4% on average and save up to 52.1% of the L1D power consumption. As our future work, we will analysis the reasons on request fails at L1 data cache and then apply appropriate information to improve the proposed technique for high performance GPUs.

REFERENCES

[1] W. Jia, K. Shaw and M. Martonosi, "MRPB: Memory Request Prioritization for Massively Parallel Processors," in the IEEE International Symposium on High Performance Computer Architecture (HPCA), pp272-283, 2014.

[2] NVIDIA, "Whitepaper: NVIDIA's Next Generation CUDA Compute and Graphics Architecture: Fermi," 2009.

[3] Y. Torres and A. Escribano, "Understanding the Impact of CUDA Tuning Techniques for Fermi," In High Performance Computing and Simulation (HPCS), pp. 631-639, 2011.

[4] A. Jog, O. Kayiran, N. Nachiappan, A. Mishra, M. Kandermir, O. Mutlu, R. Iyer and C. Das, "OWL: Cooperative Thread Array Aware Scheduling Techniques for Improving GPGPU Performance," in the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), pp. 395-406, 2013.

[5] S. Lee, A. Arunkumar and C. Wu, "CAWA: Coordinated Warp Scheduling and Cache Prioritization for Critical Warp Acceleration of GPGPU Workloads," in the International Symposium on Computer Architecture (ISCA), pp. 515-527, 2015.

[6] M. Lee, G. Kim, J. Kim, W. Seo, Y. Cho and S. Ryu., "iPAWS: Instruction-Issue Pattern-based Adaptive Warp Scheduling for GPGPUs," in the IEEE International Symposium on High Performance Computer Architecture (HPCA), pp. 370-381, 2016.

[7] V. Narasiman, M. Shebanow, C. Lee, R. Miftakhutdinov, O. Mutlu and Y. Patt, "Improving GPU Performance via Large Warps and Two-Level Warp Scheduling," in the IEEE/ACM International Symposium on Microarchitecture (MICRO), pp. 308-317, 2014.

[8] M. Gebhart, R. Johnson, D. Tarjan, S. Keckler, W. Dally, E. Lindoholm and K. Skadron, "Energy-efficient Mechanisms for Managing Thread Context in Throughput Processors," in the International Symposium on Computer Architecture (ISCA), pp. 235-246, 2011.

[9] W. Fung, I. Sham, G. Yuan and T. Aamodt, "Dynamic Warp Formation and Scheduling for Efficient GPU Control Flow," in the IEEE/ACM International Symposium on Microarchitecture (MICRO), pp. 407-420, 2007.

[10] M. Qureshi, A. Jaleel, Y. Patt, S. Steely and J. Emer, "Adaptive Insertion Policies for High Performance Caching," in the International Symposium on Computer Architecture (ISCA), pp. 381-391, 2007.

[11] C. T. Do, H. J. Choi, J. M. Kim and C. H. Kim, "A New Cache Replacement Algorithm for Last-Level Caches by Exploiting Tag-Distance Correlation of Cache Lines," in Microprocessors and Microsystems, 39(4), pp. 286-295, 2015.

[12] A. S. Leon, B. Langley and J. L. Shin, "The UltraSPARC T1 Processor: CMT Reliability," In Custom Integrated Circuits Conference, pp. 555-562, 2006.

[13] T. Rogers, M. O'Connor and T. Aamodt, "Cache-consciou s Wavefront Scheduling," in the IEEE/ACM International Symposium on Microarchitecture (MICRO), pp. 72-83, 2012.

[14] NVIDIA, "NVIDA Tegra Multiprocessor Architecture," 2010.

[15] Y. Wu, R. Rakvic, L. Chen, C. Miao, G. Chrysos and J. Fang, "Compiler Managed Micro-cache Bypassing

www.kci.go.kr

for High Performance EPIC Processors," in the IEEE/ACM International Symposium on Microarchitecture (MICRO), pp. 134-145, 2002.
[16] T. L. Johnson and W.-M. W. Hwu, "Run-time Adaptive Cache Hierarchy Management via Reference Analysis," in the International Symposium on Computer Architecture (ISCA), pp. 315-326, 1997.
[17] M. Kharbutli and D. Solihin, "Counter-based Cache Replacement and Bypassing Algorithms," in IEEE Transactions on Computers, 57(4), pp. 433-447, 2008.
[18] A. Bakhola, G. Yuan, W. Fung, H. Wong and T. Aamodt, "Analyzing CUDA Workloads Using a Detailed GPU Simulator," in the International Symposium on Analysis of Systems and Software (ISPASS), pp.163-174, 2009.
[19] H. Liu, M. Ferdman, J. Huh, and D. Burger, "Cache Bursts: A New Approach for Eliminating Dead Blocks and Increasing Cache Efficiency," in the IEEE/ACM International Symposium on Microarchitecture (MICRO), pp. 222-233, 2008.
[20] D. Kirk and W. Hwu, "Programming Massively Parallel Processors," 2010.
[21] C. Wu, A. Jaleel, W. Hasenplaugh, M. Martonosi, S. Steely Jr. and J. Emer, "SHiP: Signature-based Hit Predictor for High Performance Caching," in the IEEE/ACM International Symposium on Microarchitecture (MICRO), pp. 430-441, 2011.
[22] NVIDA, CUDA SDK http://developer.nvidia.com/gpu-computing-sdk.
[23] X. Chen, L. Chang, C. Rodrigues, J. Lv, Z. Wang, and W. Hwu, "Adaptive Cache Management for Energy-Efficient GPU Computing," in the IEEE/ACM International Symposium on Microarchitecture (MICRO), pp.343-355, 2014.
[24] N. Duong, D. Zhao, T. Kim, R. Cammarota, M. Valero, and A. Veidenbaum, "Improving Cache Management Policies Using Dynamic Reuse Distances," in the IEEE/ACM International Symposium on Microarchitecture (MICRO), pp. 389-400, 2012.
[25] X. Xie, Y. Liang, Y. Wang, G. Sun and T. Wang, "Coordinated Static and Dynamic Cache Bypassing for GPUs," in the IEEE International Symposium on High Performance Computer Architecture (HPCA), pp. 76-88, 2015.
[26] S. Che, M. Boyer, J. Meng, D. Tarjan, J. Sheaffer, S.-H. Lee and K. Skadron, "Rodinia: A Benchmark Suite for Heterogeneous Computing, "intheIEEEInternationalSymposiumonWorkloadCharacterization,(IISWC), pp. 44-54, 2009.
[27] S. Hong and H. Kim, "An Integrated GPU Power and Performance Model," in the International Symposium on Computer Architecture (ISCA), pp. 280-289, 2010.
[28] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N. Kim, T. Aamodt and V. Reddi, "GPUWattch: Enabling Energy Optimizations in GPGPUs," in the International Symposium on Computer Architecture (ISCA), pp. 487-498, 2013.

Authors

Thuan Cong Do received the B.S. degree in Electronics and Telecommunications from Hanoi University of Science and Technology, Hanoi, Vietnam in 2012, the M.S. and Ph.D. degrees from Chonnam National University, Gwangju, Korea in 2014 and 2018, respectively. He is currently a research professor in the Department of Computer Science, Korea University, Seoul, Korea. His research interests include computer architecture, parallel processing, microprocessors, embedded systems, and GPGPU.

Gwang Bok Kim received the B.S degree and M.S in electronics and computer engineering from Chonnam National University, Gwangju, Korea in 2013 and 2015 respectively.He is currently a Ph.D student at Chonnam National University. His research interests include computer architecture, low power systems, and GPGPU.

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 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.

www.kci.go.kr