Evaluating GPGPU Memory Performance Through the C-AMAT Model

Ning Zhang
Illinois Institute of Technology
Chicago, IL 60616
USA
nzhang23@hawk.iit.edu

Chuntao Jiang
Foshan University
Foshan, Guangdong 510000
China
chuntjiang@gmail.com

Xian-He Sun
Illinois Institute of Technology
Chicago, IL 60616
USA
sun@iit.edu

Shuaiwen (Leon) Song
Pacific Northwest National Lab
Richland, WA 99354
USA
shuaiwen.song@pnnl.gov

ABSTRACT

General Purpose Graphics Processing Units (GPGPU) have become a popular platform to accelerate high performance applications. Although they provide exceptional computing power, GPGPU impose significant pressure on the off-chip memory system. Evaluating, understanding, and improving GPGPU data access delay has become an important research topic in high-performance computing. In this study, we utilize the newly proposed GPGPU/C-AMAT (Concurrent Average Memory Access Time) model to quantitatively evaluate GPGPU memory performance. Specifically, we extend the current C-AMAT model to include a GPGPU-specific modeling component and then provide its evaluation results.

CCS CONCEPTS

• Computer systems organization → Architectures → Parallel architectures → Single instruction, multiple data

KEYWORDS

GPGPU, Memory Performance Evaluation, C-AMAT

ACM Reference format:
N. Zhang, C. Jiang, X. Sun, and S. Song. 2017. Evaluating GPGPU Memory Performance Through the C-AMAT Model. In *Proceedings of ACM SIGHPC MCHPC 2017, 1st International Workshop on Memory Centric Programming for HPC, in conjunction with SC'17, Denver, CO USA, November 2017 (MCHPC'17)*, 5 pages.
DOI: 10.1145/3145617.3158214

1 INTRODUCTION

General Purpose Graphics Processing Units (GPGPU) are increasingly important for high performance computing (HPC). They have been widely used as accelerators for general purpose computing. However, they suffer from performance variability on non-graphic applications, and often provide single digit utilization for scientific applications due to inefficient data access. To mitigate the data access delay, understanding GPGPU memory performance and its contributing factors is essential for efficiency optimization. Since data access delay is typically application-dependent, the knowledge about such delay is often difficult to obtain due to GPGPU' massive parallelism. Currently, using GPGPU efficiently in scientific computing is labor intensive as well as a trial and error process.

We apply the C-AMAT (Concurrent Average Memory Access Time) [1] model to GPGPU for a better understanding of their memory system performance. Our GPGPU/C-AMAT model considers warp-level data access as the basic unit operation because warps are executed in a SIMD fashion. In other words, a warp will stall if any of its threads stalls. By considering both data access concurrency and locality, the GPGPU/C-AMAT model provides a reliable metric that can accurately reflect the performance of GPGPU memory system.

The rest of this paper is organized as follows. Section 2 introduces the background which includes the basic C-AMAT definition and formulation in GPGPU architecture. Section 3 introduces the evaluation methodology of C-AMAT for GPGPU. In Section 4, we provide the evaluation details. Then, Section 5 presents related work. Finally, Section 6 gives the conclusion.

2 BACKGROUND

In this section, we first introduce the GPGPU architecture and scheduling schemes. Then, we introduce the C-AMAT model and its interpretation toward GPGPU.

2.1 The GPGPU Architecture

A GPGPU consists of many simple in-order cores, which are typically organized under the "single-instruction, multiple-threads" (SIMT) execution model with lanes of 8 to 32 [2][3]. As shown in Fig. 1, a GPGPU has several SM (streaming multiprocessor) cores, each SM core consists of 32 or 64 small

MCHPC'17, November 12–17, 2017, Denver, CO, USA

ACM ISBN 978-1-4503-5131-7/17/11…$15.00
https://doi.org/10.1145/3145617.3158214

cores, and several load/store units. Each SM core has a private L1 data cache, a read-only texture cache and a constant cache, along with a low-latency shared memory. SM cores have direct access to L2 cache and are clustered for organization. They have a share Memory Controllers (MC) to access DRAM. The GPGPU architecture employs the memory coalescing technique, where nearby memory accesses are coalesced into a single cache line, so that the total number of memory requests is reduced.

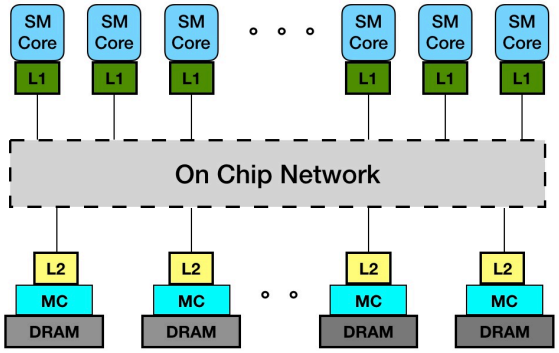

Figure 1: Hardware Architecture of GPGPU

Fig. 2 shows the software hierarchy of a GPGPU application consisting of threads, warps, CTAs (Cooperative Thread Arrays), and kernels. A group of threads constitute a "CTA" or "thread block". A CTA is essentially a batch of threads that can coordinate among each other by synchronizing their execution streams using barrier instructions. Since all the synchronization primitives are encapsulated in the CTA, execution of CTAs can be performed in any order [4][5]. This helps in maximizing the available parallelism and any core is free to schedule any CTA. Further, each kernel is associated with many CTAs, and one or multiple kernels form a GPGPU application. A "warp" or a "wavefront" is the granularity at which threads are scheduled to the pipeline, and is a group of 32 threads. Note that a warp is an architectural structure rather than a programming model concept.

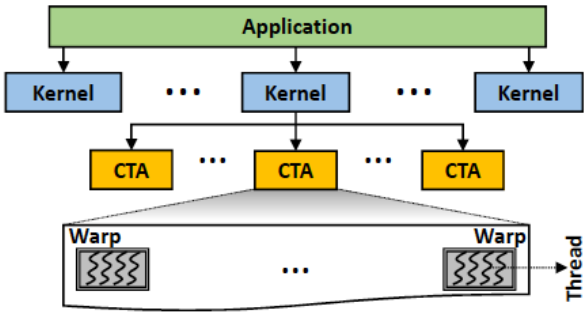

Figure 2: Software Architecture of GPGPU

GPGPU performances are highly influenced by scheduling [6][7][8]. Scheduling is typically a three-step process. First, a kernel of an application is launched on the GPGPU. Generally, only one kernel is active at a time. After launching the kernel, the second step is that the global block (CTA) scheduler assigns CTAs of the launched kernel to all the available cores. The CTA assignment is done in a load-balanced round-robin fashion. If there are enough CTAs, each core is assigned with at least one CTA. Then, if a core can execute multiple CTAs, a second round of assignment starts; if there are enough available CTAs, this process continues until all CTAs have been assigned or all the cores have been assigned with their maximum limit of CTAs. Assuming there are enough CTAs to schedule, and the number of concurrently executing CTAs in the system is equal to N. The maximum CTAs (N) per-core is limited by core resources. Given a baseline architecture, CTAs(N) may vary for a given kernel depending on the resources needed by the CTAs. After the CTA assignment, the third step is to schedule the warps, which are associated with the launched CTA(s), on a core. The warps are typically scheduled in different scheduling policies such as round-robin and GTO (Greedy Then Oldest) to the SIMT lanes. Note that warp is the smallest unit that can be scheduled on GPGPU [9][10].

2.2 C-AMAT and its Interpretation on GPGPU

The C-AMAT (Concurrent-AMAT) memory performance model [1] is an extension of the traditional AMAT (Average Memory Access Time) model [11] to consider concurrent data access. Quantitatively speaking, C-AMAT is equal to the total memory access cycles divided by the total number of memory accesses [12]. Let $T_{MemCycle}$ represent the total number of cycles executed in which there is at least one outstanding memory reference; and let C_{MemAcc} represent the total number of memory accesses, then we have,

$$C - AMAT = \frac{T_{MemCycle}}{C_{MemAcc}} \quad (1)$$

In GPGPU, a warp cannot progress until all its threads have fetched their data. In other words, if one thread has a cache miss, then the while warp will be stalled. Therefore, different with CPU, the C-AMAT model for GPGPU considers warp-level data accesses instead of counting thread-level access. Therefore, in the two-parameter C-AMAT definition Eq. (1), for GPGPU, C_{MemAcc} represents the total number of warp-level memory accesses. The $T_{MemCycle}$ calculation for GPGPU is the same as that of CPU without any change. An overlapping mode is adopted for counting memory access cycles. That is when more than one warps access memory at the same memory cycle, $T_{MemCycle}$ only increases by one. Another important feature of $T_{MemCycle}$ is that it only counts memory active cycles. That is if the memory is idle (here idle means there is no ongoing data access under its memory branch), then there is no $T_{MemCycle}$ counting.

A five-parameter form of C-AMAT is shown in Equation (2) [1], which is extremely useful for performance analysis.

$$C - AMAT = \frac{H}{C_H} + pMR \times \frac{pAMP}{C_M} \quad (2)$$

In Eq. (2), the first parameter C_H represents the hit concurrency; the second parameter C_M represents the pure miss concurrency. In modern cache design, multi-port cache, multi-banked cache or pipelined cache structures could contribute to C_H. Non-blocking cache structure could contribute the C_M. *pMR* (Pure Miss Ratio) in Eq. (2) is the number of pure misses over the total number of accesses. Pure miss is an important concept introduced by C-AMAT [1]. It means that the miss contains at least one miss cycle which does not have any hit access. *pAMP* (Average Pure Miss Penalty) is the average number of pure miss cycles per pure miss

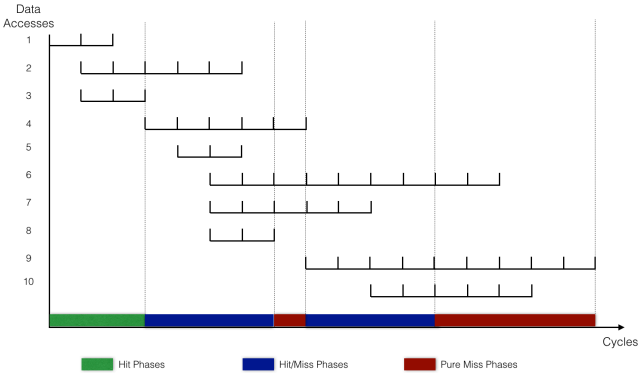

Figure 3: A C-AMAT Example

access. Fig. 3 provides a demonstration example to illustrate the "hit", "miss", and "pure miss" cycle concept. There are 10 different memory accesses in Fig. 3. Each access contains 2 cycles for cache hit operations. If it is a miss, additional miss penalty cycles will be required. The number of miss penalty cycles is uncertain because the miss will happen in different level caches. From Fig. 1, access 1, 3, 5, 8 are hit accesses and access 2, 4, 6, 7, 9, 10 are miss accesses. Access 2, access 4, access 7 and access 10 both have 3-cycle miss penalty while both access 6 and 9 have 7-cycle miss penalty. When considering the access concurrency and the definition of pure miss, access 4 and 7 contain 1 pure miss cycle; access 6 has 3 pure miss cycles; access 9 has 5 pure miss cycles; access 10 has 3 pure miss cycles. There are three kinds of phases in Fig. 3 which are Hit Phases, Hit/Miss Phases and Pure Miss Phases. In Hit Phases, there are only hit cycles. In Pure Miss Phases, there are only pure miss cycles and in Hit/Miss Phases both miss cycles and hit cycles exist. The overlapping of hit and miss access happens in Hit/Miss Phases. According to Eq. (1), C-AMAT in Fig. 3 is 17 cycles out of 10 accesses or 1.7 cycle per access. Another approach to calculate C-AMAT is to use hit and miss concurrency factors. The critical question is how to obtain an accurate average C_H and C_M. Here a weighted method is applied to calculate the average value in [1]. As the example shown in Fig. 3, there are 20 hits over 11 memory cycles. Therefore, C_H = 20/11 = 1.82. And there are two pure miss phases, with totally 13 pure misses on 6 cycles. Therefore, C_M = 13/6 = 2.17; pAMP = 13/5 =2.6; pMR = 5/10. Thus Eq. (2) is equal to

$$\frac{H}{C_H} + pMR \times \frac{pAMP}{C_M} = \frac{2}{20/11} + \frac{5}{10} \times \frac{13/5}{13/6} = 1.7$$

The C-AMAT values calculated by Eq. (1) and Eq. (2) are the same because these two equations are equivalent.

In GPGPU, convert the data access at the warp level, the C-AMAT parameters in Eq. (2) have their new meaning as follows.

- Warp-level cache hit (H): when all threads of the warp have a cache hit.
- Warp-level cache miss (M): when one or more threads of the warp has a cache miss.
- Warp-level cache pure miss (pM): all warps cannot be scheduled due to warp-level miss.
- Average Warp-level hit concurrency (C_H): accumulated warp-level hit cycles / active warp-level hit cycles.
- Average Warp-level pure miss concurrency (C_M): accumulated warp-level pure miss cycles / active warp-level pure miss cycles.

Under the wrap-level definitions, the GPGPU/C-AMAT model is appropriate for GPGPU for the following reasons. Firstly, miss and hit are defined at the warp-level. This is because warp is executed under the SIMD execution model. One miss will stall the whole wrap. Secondly, the pure miss concept introduced by the C-AMAT model is also extended to the warp-level in GPGPU. This is because GPGPU employs warp-level scheduling to increase throughput and hiding data access latency. When one warp is stalled due to miss, another warp in the waiting queue will be scheduled into the pipeline for execution. This miss/fetch process will continue until the waiting queue is empty. Therefore, a GPGPU/C-AMAT pure miss is defined as all warps are stalled by data misses. The warp-level pure misses of GPGPU/C-AMAT are the misses causing a SM core idle. They provide a reliable measurement that can accurately reflect the runtime data transfer behaviors of GPGPU memory systems.

3 GPGPU/C-AMAT MEASUREMENT

To measure the C-AMAT value of the memory system in GPGPU, only two parameters need to be measured as given by Eq. (1). They are C_{MemAcc} (Memory access) and $T_{MemCycle}$ (Memory Active Cycles). To precisely measure these two parameters, a C-AMAT Measurement Structure is designed in Fig. 4, which shows the design logic of measuring the C-AMAT of L1 Dcache in each SM core of GPGPU.

There are several components on Fig.4, namely MSHR (Miss Status Holding Register), Load/Store Unit and C-AMAT Measurement Logic (CML), etc. MSHR is a structured table. It records cache miss information, such as access type (load/store), access address, and return register. When the MSHR table is empty, there is no any outstanding cache miss. While the MSHR table is not empty, it means the cache is actively waiting for data return from the next level layer of the memory hierarchy. The Load/Store Unit is a specialized execution unit responsible for executing all load and store instructions and loading data from memory or storing data back to memory from registers. When the Load/Store Unit is busy, it means the L1 Dcache is also active because Load/Store Unit is sending data access requests to L1

Dcache or waiting for the response from L1 Dcache. So CML can

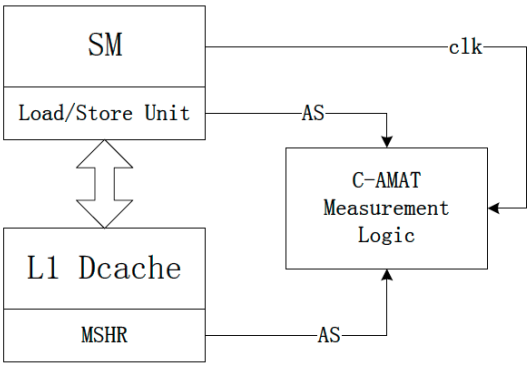

Figure 4: C-AMAT Measurement Structure

count the L1 Dcache active cycles through AS (Active Signal) from Load/Store Unit and MSHR. One SM have many Load/Store Units and if only one of Load/Store Units is busy, the AS will be valid. We set CML's clock be the same as that of SM to make time synchronized to count the L1 Dcache active cycles. The active cycle counting logic of L1 Dcache is shown in Table 1.

Table 1: L1 Dcache Active Cycle Counting Logic

Pseudo Code for Counting L1 Dcache Active Cycles

If (MSHR table is not empty) // having pending cache miss
  Active_cycle ++;
Else if (Load/Store Unit is busy) //cache is accessing
  Active_cycle ++;
Else
  Active_cycle does not change

The CML only needs two long-bit sized registers (e.g., 64-bit register is sufficient for most computer systems) and some detecting logic. One register counts the total number of memory access cycles; the other counts memory accesses. While a memory accesses counter is already provided by existing GPGPU performance counters, the CML just reads this count and does the C-AMAT calculation with these two counters as Eq. (1) shows.

4 EXPERIMENTAL SETUP AND RESULTS

A detailed simulation model of GPGPU in GPGPU-Sim simulator was adopted, which simulates NVIDIA's Fermi and GT200 architectures. This simulator also includes an integrated and validated energy model, GPUWattch. The intention of GPGPU-Sim is to provide a substrate for architecture research rather than to exactly model any commercial GPU.

In our experiments, we use GPGPU-Sim version 3.1.0, a cycle-level version that focuses on "GPU computing" (general purpose computation on GPU). Table 2 shows the details of the key simulation configuration parameters and their values. We simulate GPGPU architecture with 15 processing clusters, and 1 SM core per cluster. The employed warp scheduler uses GTO (Greedy Then Oldest) scheduling policy. There are 16 register banks. The number of warp scheduler per core is one. In one SM core, the maximum number of concurrent CTAs is 8. In fact, we configure the parameters in our experiment following the Fermi microarchitecture developed by NVIDIA. We deploy our experiments on this mature architecture to get more reasonable results.

Table 2: GPGPU-Sim Simulation Configuration Parameters

Parameter	Value
Processing clusters	15
SM core per cluster	1
Warp scheduler	GTO
Number of register banks	16
Number of warp schedulers per core	2
Maximum number of concurrent CTAs per SM	8

The C-AMAT value for L1 Dcache in simulated GPGPU architecture is measured in our experiments. The two-parameter C-AMAT measurement methodology is employed, which measures $T_{MemCycle}$ and C_{MemAcc}. Fig. 5 and Fig. 6 show the L1 C-AMAT values of 15 SM cores for B+Tree and Particular Filter applications, respectively. We can observe two phenomena through the figures. One phenomenon is that L1 Dcache C-AMAT values vary largely for different CUDA applications. For example, the average L1 cache C-AMAT value is about 10 for B+Tree, but for Particular Filter it is about 40 which means the L1 Dcahe performance of Particular Filter is four times worse than that of B+Tree. It is because B+Tree is cache friendly application while Particular Filter is cache unfriendly application. This implies that

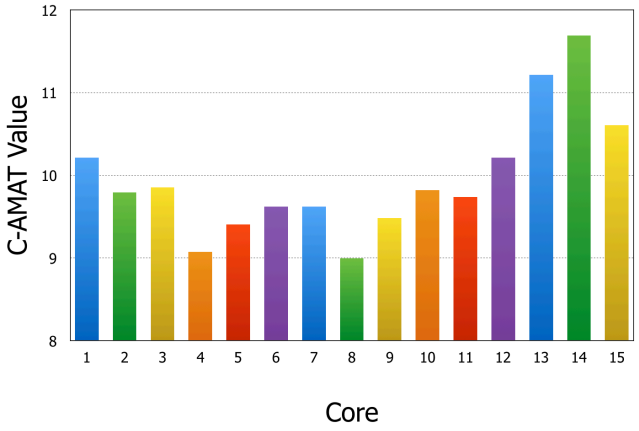

Figure 5: L1 Dcache C-AMAT values of 15 SM cores for B+Tree

different applications have different types of data requests, leading to totally different data access pressure on GPGPU memory hierarchy (L1 Dcache in this example). Therefore, we need to adjust the data requests and the data supply to make them match each other, to achieve an optimized performance. Another

phenomenon we have observed is that even for the same application, the L1 cache C-AMAT value for different SM cores are also different. For example, in B+Tree application, the smallest C-AMAT value is about 9, but the biggest C-AMAT value is about 11.5; in the Particular Filter application, the smallest C-AMAT value is about 38, but the biggest C-AMAT value is up to 48. This implies that different SM cores have different data requests. Therefore, the optimization work is also needed at SM core level to balance the workloads between SM cores. Please notice here the balance is on data accesses, which is not necessarily the same as operation count.

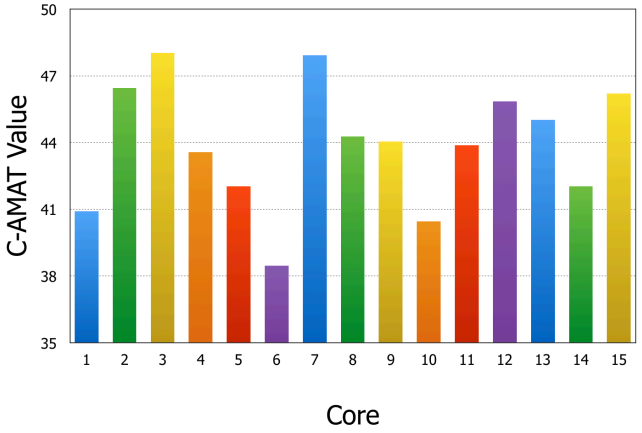

Figure 6: L1 Dcache C-AMAT values of 15 SM cores for Particular Filter

Through the experimental results, we can see that the GPGPU/C-AMAT model can accurately indicate the performance of GPGPU memory systems and data access behaviors. An unbalanced data access behavior at warp level will hinder applications from fully utilizing the GPGPU computing power. In the future, we plan to apply our GPGPU/C-AMAT measurement to guide GPGPU memory system optimizations, e.g. to balance data access requests among SM cores via CTA scheduling.

5 RELATED WORK

Sun et al. [1] firstly proposed C-AMAT for modeling memory systems in a general CPU environment. Govindaraju et al. [13] presented a memory model to improve the performance of applications (SGEMM, FFT) by improving texture cache usage. The work by Liu et al [14] modeled the performance of bio-sequence alignment applications written in GLSL (OpenGL Shading Language) [15]. Compared with the C-AMAT model in GPGPU, these models are simplified for specific applications where the C-AMAT model in GPGPU is generic to all GPGPU applications.

6 CONCLUSIONS

In this paper we have applied the general C-AMAT Model on GPGPU memory systems. Based on the characteristics of GPGPU, the parameters of C-AMAT are redefined for the application domain. We have also designed a C-AMAT measurement methodology for GPGPU. Based on the design of measurement, the C-AMAT values of the L1 Dcache of GPGPU are measured and studied. The modern GPGPU system simulator, GPGPU-Sim, is enhanced for the C-AMAT measurement. Through experimental results, we can see that the L1 Dcache performances of GPGPU are very different among different applications and among different SM cores, which lead to future performance optimizations for GPGPU memory systems.

ACKNOWLEDGMENTS

This work is supported in part by the National Science Foundation, under grant CCF-1536079, CNS-1338078, and the Chameleon Cloud.

REFERENCES

[1] X.-H. Sun and D. Wang, "Concurrent Average Memory Access Time," Computer, vol. 47, no. 5, pp. 74–80, 2014.

[2] A. Jog, O. Kayiran, N.C. Nachiappan, and et al. "OWL: Cooperative Thread Array Aware Scheduling Techniques for Improving GPGPU Performance," SIGPLAN Vol.48, pp. 395-406, 2013.

[3] M. Lee et al., "Improving GPGPU Resource Utilization through Alternative Thread Block Scheduling," IEEE 20th International Symposium on High Performance Computer Architecture (HPCA), Orlando, FL, pp. 260-271, 2014.

[4] P. Xiang, Y. Yang, and H. Zhou, "Warp-level divergence in gpus: Characterization, impact, and mitigation," in Proceedings of 2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA-20), 2014.

[5] O. Kayıran, A. Jog, M. T. Kandemir, and C. R. Das, "Neither more nor less: Optimizing thread-level parallelism for gpgpus," in Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques (PACT'13), 2013.

[6] M. Lee, S. Song, J. Moon, J. Kim, W. Seo, Y. Cho, and S. Ryu, "Improving gpgpu resource utilization through alternative thread block scheduling," in Proceedings of 2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA-20), 2014.

[7] S.-Y. Lee, A. Arunkumar, and C.-J. Wu, "Cawa: coordinated warp scheduling and cache prioritization for critical warp acceleration of gpgpu workloads," in Proceedings of the 42nd Annual International Symposium on Computer Architecture (ISCA-42), 2015.

[8] J. Wang and S. Yalamanchili, "Characterization and analysis of dynamic parallelism in unstructured gpu applications," in Proceedings of 2014 IEEE International Symposium on Workload Characterization (IISWC'14), 2014.

[9] J. Wang, N. Rubin, A. Sidelnik, and S. Yalamanchili, "Dynamic thread block launch: A lightweight execution mechanism to support irregular applications on gpus," in Proceedings of the 42nd Annual International Symposium on Computer Architecuture (ISCA-42), 2015.

[10] G. Chen and X. Shen, "Free launch: Optimizing gpu dynamic kernel launches through thread reuse," in Proceedings of the 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-48), 2015.

[11] Wulf, W. A. and Mckee, S. A. 1995. Hitting the Memory Wall: Implications of the Obvious. ACM SIGARCH computer architecture news 23, 1, 20-24.

[12] X.-H. Sun, "Concurrent-AMAT: A Mathematical Model for Big Data access," HPC Magazine, 2014.

[13] N. K. Govindaraju, S. Larsen, J. Gray, and D. Manocha. A memory model for scientific algorithms on graphics processors. In SC, 2006

[14] S. B. Weiguo Liu, Muller-Wittig. Performance predictions for general-purpose computation on gpus. 2007.

[15] J. Kessenich, D. Baldwin, and R. Rost. The OpenGL shading language. http://www.opengl.org/documentation.