A Performance Prediction Model for Memory-intensive GPU Kernels

Zhidan Hu
College of Computer
National University of Defense Technology
Hunan, China
Email: huzd@nscc-tj-gov.cn

Guangming Liu
College of Computer
National University of Defense Technology
Hunan, China
Email: liugm@nscc-tj-gov.cn

Zhidan Hu
College of Computer
National University of Defense Technology
Hunan, China
Email: dongwr@nscc-tj-gov.cn

***Abstract*—Commodity graphic processing units (GPUs) have rapidly evolved to become high performance accelerators for data-parallel computing through a large array of processing cores and the CUDA programming model with a C-like interface. However, optimizing an application for maximum performance based on the GPU architecture is not a trivial task for the tremendous change from conventional multi-core to the many-core architectures. Besides, the GPU vendors do not disclose much detail about the characteristics of the GPU's architecture.**

To provide insights into the performance of memory-intensive kernels, we propose a pipelined global memory model to incorporate the most critical global memory performance related factor, uncoalesced memory access pattern, and provide a basis for predicting performance of memory-intensive kernels. As we will demonstrate, the pipeline throughput is dynamic and sensitive to the memory access patterns. We validated our model on the NVIDIA GPUs using CUDA (Compute Unified Device Architecture). The experiment results show that the pipeline captures performance factors related to global memory and is able to estimate the performance for memory-intensive GPU kernels via the proposed model.

Keywords—GPU; CUDA; performance prediction; memory-intensive

I. INTRODUCTION

Modern parallel processing architecture has been switched from conventional multi-core into the many-core parallel architecture. As a representative of the many-core architecture, the emergence of low cost programmable GPU computing substrates from NVIDIA, Intel, and AMD have made data parallel architectures a promising alternative building block for high performance heterogeneous systems. The dominant programming model involves the use of bulk-synchronous-parallel computing models embodied by languages such as CUDA, OpenCL, and DirectX Compute. These data-parallel languages implement single instruction multiple thread (SIMT) models of computation that specify a large number of data-parallel threads that can be readily exploited by hardware multi-threading and single instruction multiple data (SIMD) cores.

The GPU architecture is confirmed to be suitable for data-parallel and compute-intensive applications can maximize the computing capability as memory operations often degrade performance heavily, especially off-chip memory accesses. The memory performance is sensitive to the memory access pattern which describes how data indexes are referred by consecutive threads within a warp. Ideally, all the memory accesses within a warp can be combined into one memory transaction only if all the referred addresses fall within one memory segment and the address align requirement is satisfied, named coalesced memory access. Otherwise, multiple memory transactions will be triggered to serve a memory instruction of one warp, which is called uncoalesced memory access. The following computations that depend on the required data will have to wait for an even longer time.

To provide insight into how global memory performance is affected by memory access pattern and how application performance is affected by memory performance, we propose a pipelined model for global memory accesses, and further estimate execution time for memory-intensive GPU kernels using the pipeline throughput. The model can be used statically without even executing an application. The basic intuition of our model is that estimating the cost of memory operations is the key component of estimating the performance of memory-intensive parallel GPU applications.

We evaluate our pipeline global memory model and the pipeline throughput based memory performance prediction method on the CUDA programming language with a C-like interface. We compare the results of our predicted results with the actual execution time for several memory-intensive kernels and show that the prediction error is acceptable.

The contributions of our work can be concluded as follows:

- We investigate the process of global memory transaction servicing and figure out that multiple transactions can be processed in parallel. A pipeline model is constructed to represent the stage-by-stage process of each memory transaction and describe how multiple memory transactions are serviced concurrently.
- Since memory-intensive kernel performance is dominated by the efficiency of memory operations, we utilize the pipeline throughput to predict the execution time of several memory-intensive GPU kernels.

II. RELATED WORK

General purpose graphics processing units (GPGPUs) have gained a surge of interests among the academic and industrial community due to its outstanding computing power, economical and energy efficiency[1].The performance prediction for GPU programs tends to be difficult because of GPU-specific performance factors such as shared memory bank conflict, control flow divergence, and uncoalesced memory access, etc. The global memory often acts as the performance bottleneck even for most data-parallel GPU kernels [2-8].

Several studies have been devoted to estimate the memory performance of GPU applications. Baghsorkhi [9] presented a software-based approach for monitoring the memory hierarchy performance in GPUs, but their work emphasized on the measure methodology. Kim and Shrivastava [10] proposed a tool, CuMAPz, to help programmers explore different ways of using memory hierarchy in the GPUs. They focused on the problem of improving application performance by using shared memory, while we tend to model the impact of global memory. The global memory model, proposed by Hong and Kim [11], shares the most common with us that uncoalesced memory accesses can be synthesized by coalesced accesses, but they did not explain how the global memory handles those accesses and model the memory performance in particular.

III. BACKGROUND

Commodity GPUs support both graphics and general computing which is referred to as GPGPU (General Purpose Graphics Process Unit) where the GPU is viewed as a massively multi-threaded architecture containing hundreds of processing elements. These processing cores are organized as a two-layered hierarchy. As shown in Fig. 1, the top level is combined with a set of SIMD fashion Streaming Multiprocessors (SMs), each containing multiple Streaming Processors (SPs). For NVIDIA Tesla M2050, there are 14 SMs and 32 SPs for each SM, which makes for a total of 448 processing cores. All SMs are connected to an off-chip GDDR via a interconnection network and each SM has its private on-chip memories, which is consists of shared memory, constant memory, texture memory, registers, and other caches.

The CUDA programming model employs a SPMD model and the CUDA API allows the programmer to create large number of threads to execute the data-parallel code parts of a CPU program as GPU kernels. Threads are also grouped into blocks and blocks make up a kernel grid. Blocks are scheduled to run on each SM in a round-robin manner and multiple blocks can be mapped to an identical SM simultaneously but one block can only be mapped to a single SM. Threads within a block can communicate with each other throughput the block-private shared memory and communication between blocks can only be done throughput the global memory. The actual basic granularity of threads to be scheduled, managed, and executed is thread warp, which usually contains a constant size of threads. In order to maximize the efficiency of computations, threads within a warp should execute along the identical control flow path, as threads follow different paths need to be serially executed until all threads in the warp reach the next re-converge point. Also, memory access requests are issued per (half) warps, and one memory transaction will satisfy a warp's execution of one memory instruction in the coalesced situation.

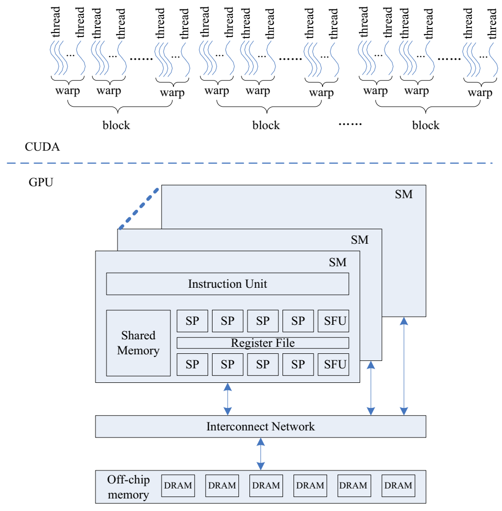


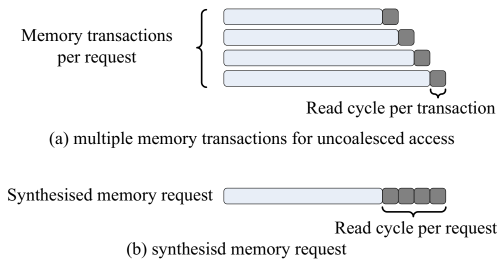
Fig. 1 Overview of GPU architecture and CUDA

IV. PIPELINE FOR GLOBAL MEMORY ACCESS

The concept of memory throughput is mostly referred to data throughput of DRAM in the spectrum of GPU applications. However, as a throughput-oriented architecture, the efficiency of memory request handling is more critical to memory performance than the latency of each memory request, just like the SIMD pipeline in the stream-multiprocessors in GPUs. In this work, we extend memory throughput to represent the speed of servicing memory requests from all threads at the warp level parallelism.

A. Synthesized global memory transaction

The efficiency of global memory is largely affected by the memory access patterns, which may cause multiple memory transactions for one memory request. The global memory access has a latency of hundreds cycles, which is a round trip time to the DRAM that includes the DRAM access time and the address and data transfer time. We assume the DRAM access has the longest latency among all the subtle steps of the memory transaction processing. Then all transactions per one request can be synthesized by one transaction with a lengthened DRAM access latency as depicted in Fig. 2.


Memory transactions per request
Read cycle per transaction
(a) multiple memory transactions for uncoalesced access
Synthesised memory request
Read cycle per request
(b) synthesisd memory request
Fig. 2 Synthesis memory transactions of uncoalesced memory request

The value of DRAM access for a memory transaction can be obtained from the hardware specification of DRAM manufacture. For Tesla M2050 [12], the read cycle per each transaction rc after parallelized by p memory partitions is

$$\mathrm{rc} = \frac{t_{RAS} + t_{RP}}{p \times f_{mem}} = 1.294\ ns$$

where f_{mem} stands for the clock frequency of DRAM chips, and t_{RAS} and t_{RP} are two hardware parameters from the datasheet. Assume the average number of memory transactions that caused by one memory request is tpr, then the DRAM access latency of the synthesized transaction in Fig. 2 can be calculated as

$$\mathrm{rc_{syn}} = rc \times tpr$$

B. Pipeline for global memory accesses

Based on the synthesized global memory transaction, we construct a pipeline for global memory accesses and utilize the pipeline throughput to describe the efficiency of memory requests handling. The constructed pipeline model is described from the following three aspects:

- Pipeline stage, denoted as N_p, points to the number of the most subtle steps in the memory access process. We actually do not care about its value just like the depth of the arithmetic pipeline of each SM.
- Stage length, denoted as $SL[i]$ ($0 \le i < N_p$), specifies the duration of each stage. The DRAM access has the longest latency among all the subtle steps based on the assumption presented before.
- Throughput, it actually defines the speed of memory accesses leaving the global memory system. As we all known, the throughput is constraint by the most inefficient pipeline stage, that is, the procedure of DRAM access of the synthesized memory transaction presented in the Fig. 2. The throughput t_p can be calculated as

$$\mathrm{t_p} = \frac{1}{rc_{syn}}$$

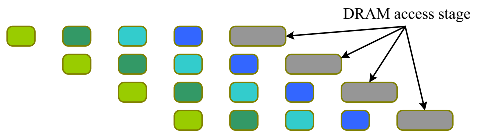


Fig. 3 A 5-stage pipeline model for global memory accesses

Figure 3 presents an example of the global memory pipeline with N_p equals to 5. For simplicity, all stages have the same latency except the last grey block, which represents the DRAM access stage. The efficiency of the pipeline is constraint by the DRAM access stage, which can be further deduced to the memory access pattern.

V. GLOBAL MEMORY PERFORMANCE MODEL

For memory-intensive kernels, performance of global memory accesses Mperf can be obtained using the following equation

$$\mathrm{M_{perf}} = \frac{m_{req}}{t_p} = 1.294 \times tpr \times w \times m_{inst}$$

where m_{req} stands for the number of global memory requests for all warps, which can be obtained via a multiplication of warp number w and average global memory requests per each warp m_{inst}. The performance model takes data size and memory access pattern as inputs and predicts the performance of memory accesses for memory-intensive kernels.

VI. EXPERIMENT RESULTS

A. Verification of the pipeline

To validate the assumption that DRAM access has the longest latency among all the subtle steps of the memory transaction processing, we collect latencies of the strided-access kernel under different number of memory requests for 4 memory access patterns, each with a corresponded value for tpr. As depicted in Fig. 4, the exact value of rc can be calculated by dividing the increment of latency by the variant of warp amount.

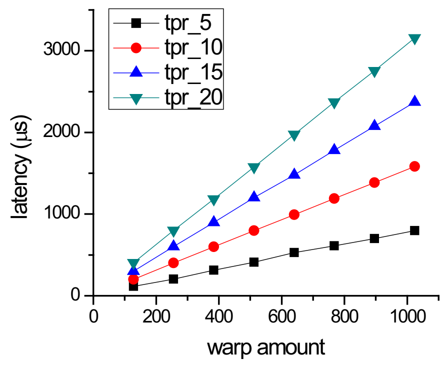


Fig. 4 Performance variation with warp amount

The deviation of DRAM access time is presented in Table I and the error is only 2.9%, which provides strong evidence to the assumption and verifies the effectiveness of the constructed pipeline.

TABLE I. VERIFICATION OF ASSUMPTION

tpr[i]	5	10	15	20
rc[i]	1.32	1.34	1.34	1.33
$\mathrm{rc_{measure}}$	1.3325			
$\mathrm{rc_{model}}$	1.294			
error	2.9%			

B. Performance prediction

1) Strided-access benchmark

Based on the verification of the pipeline, we can correlate performance with the pipeline throughput for memory-intensive kernels. Fig. 5 depicts measured execution time for strided access benchmark with respect to two different throughput values determined by stride for 64×256 and 64×512 thread configurations respectively. The variance of stride can be straight correlated with the memory access pattern, as number of generated memory transactions equals to the stride value when the stored data elements occupy 4 byte memory space. As can be seen from the figure, there is an inverse relationship between latency and throughput for both kernel configurations.

As latency of purely memory operations is not available at hand, we collect latencies with different amount of memory requests and DRAM access latency by varying the value of stride. And then we compare the variation of latency that measured with the variation calculated with the following equation

$$\Delta M_{\mathrm{perf}} = \Delta(stride \times w) \times 1.294 \times m_{inst}$$

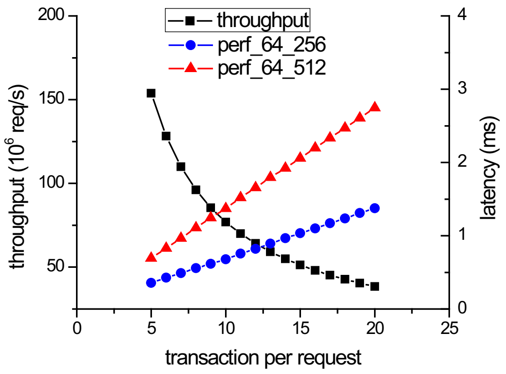


Fig. 5 Throughput vs. performance

We measure latency of the strided-access kernel for two cases as presented in Table II and calculate the error of estimating performance variation which turns out to be only 2.8%.

TABLE II. MEASURED LATENCY FOR TWO CASES

	case 1	case 2
thread configure	64×256	64×512
tran_per_req	15	10
latency (cycle)	1203170	1584332
measured variation	381162	
modeled variation	391680	
prediction error	2.8%	

The subtle prediction error indicates that global memory related performance factor, uncoalesced memory access, is captured by the proposed pipeline and the global memory performance of memory-intensive kernels can be accurately modeled via the proposed performance model.

2) *Rodinia benchmark suit*

The Rodinia benchmark suit is designed for heterogeneous computing infrastructure, and it has both OpenMP and CUDA versions [13]. The latest version of Rodinia contains 19 benchmarks covering data mining, bioinformatics, physics simulation, pattern recognition, image processing, medical imaging, graph algorithm and web mining and other domains.

From the performance domination perspective, all benchmarks can be classified into compute intensive and memory-intensive categories. According to previously published works [11], the execution time of memory-intensive GPU kernels is approximately equal to that of the memory operations. We predict performance for several kernels of three benchmarks from the Rodinia benchmark suit.

- **Compute Fluid Dynamics** (CFD) is an unstructured grid finite volume solver for the three-dimensional Euler equations for compressible flow. Effective GPU memory bandwidth is improved by reducing total global memory access and overlapping redundant computation, as well as using an appropriate numbering scheme and data layout.
- **Gaussian Elimination** computes result row by row, solving for all of the variables in a linear system. The algorithm must synchronize between iterations, but the values calculated in each iteration can be computed in parallel. The application analyses an n × n matrix and an associated 1 × n vector to solve a set of equations with n variables and n unknowns.

TABLE III. CHARACRISTICS OF MEASURED KERNELS

Kernel	**Global memory request**	**Transaction per req**
Variable_init	5	1
Compute_flux_contribution	17	1
Time_step	16	1
Fan1	2	32
Fan2	2	32

The CUDA version of CFD embodies 5 kernels and three of these are regarded as memory-intensive: variable_init, compute_flux_contributions and time_step. Although the memory access pattern is perfectly coalesced, a high memory-to-compute ratio put heavy stress on the global memory and the performance of kernels is restricted by the memory operations. There are two kernels Fan1 and Fan2 for the Gaussian Elimination benchmark. The two global memory accesses of each kernel perform the size-strided read or write and the size value is far beyond 32, which results 32 memory transactions per memory request. Table III shows the memory characteristics of these kernels and a comparison of actual run time and predicted results is presented in Table IV.

TABLE IV. PERFORMANCE PREDICTION RESULTS FOR CFD AND GAUSSIAN

Kernel	**Input size**	**Measure time(s)**	**Predicted time(s)**	**Predict error**
Variable_init	97K	0.02144	0.202036	5.6%
	193K	0.038242	0.040272	5.3%
Compute_flux_co ntributions	97K	0.076597	0.069802	10.1%
	193K	0.146076	0.136926	6.2%
Time_step	97K	0.649499	0.064755	6.4%
	193K	0.132312	0.128871	2.6%
Fan1	1024	0.002867	0.002588	9.7%
Fan2	1024	0.136716	0.131787	3.6%

3) *Stencil*

The Stencil benchmark from the Parboil benchmark is a representative of algorithms which compute values depending on the neighbouring elements. It computes an iterative Jacobi solver of the heat equation on a 3D structured grid, which can also be used as a building block for more advanced multi-grid partial differential equation (PDE) solvers.

The GPU-optimized code has one kernel, which iterate nz-2 times and the loop body issues 5 memory read and 1 memory write requests to the global memory as shared memory is disabled. We collect kernel execution time for three cases and a comparison of actual run time and predicted results is represented in Table V.

TABLE V. PERFORMANCE PREDICTION FOR STENCIL

(nx,ny,nz)	Measured time(s)	Predicted time(s)	Predict error
(512,512,4)	0.188463	0.162558	13.7%
(1024,512,4)	0.346627	0.305169	11.9%
(512,512,8)	0.466165	0.414159	11.1%

VII. CONCLUSION

In this paper, we propose a performance model particularly for global memory accesses in memory-intensive GPU kernels based on the throughput of the constructed global memory pipeline. The experiment results show that our pipeline model is capable of capturing the global memory related performance factors including uncoalesced memory access and the error of performance prediction tends to be acceptable. For GPU kernels with a high memory-access-to-compute ratio and irregular memory access patterns, the overall performance of the kernel is largely reflected on the memory performance, which can be precisely predicted via the proposed performance model.

REFERENCES

[1] NVIDIA’s next generation CUDA compute architecture: Fermi. NVIDIA Corporation, 2009.

[2] Park, In Kyu, et al. "Design and performance evaluation of image processing algorithms on GPUs." Parallel and Distributed Systems, IEEE Transactions on 22.1 (2011): 91-104.

[3] Gaster, Benedict R., and Lee Howes. "Can GPGPU Programming Be Liberated from the Data-Parallel Bottleneck." Computer 45.8 (2012): 42-52.

[4] Lee, Daren, et al. "CUDA optimization strategies for compute-bound and memory-bound neuroimaging algorithms." Computer methods and programs in biomedicine 106.3 (2012): 175-187.

[5] Wu, Hao, et al. "GPU accelerated dissipative particle dynamics with parallel cell-list updating." IEIT Journal of Adaptive & Dynamic Computing 2 (2011): 26-32.

[6] Jang, Byunghyun, et al. "Exploiting memory access patterns to improve memory performance in data-parallel architectures." Parallel and Distributed Systems, IEEE Transactions on 22.1 (2011): 105-118.

[7] Myre, Joe, et al. "Performance analysis of single-phase, multiphase, and multicomponent lattice - Boltzmann fluid flow simulations on GPU clusters." Concurrency and Computation: Practice and Experience 23.4 (2011): 332-350.

[8] Stratton, John A., et al. "Parboil: A revised benchmark suite for scientific and commercial throughput computing." Center for Reliable and High-Performance Computing (2012).

[9] Baghsorkhi, Sara S., et al. "Efficient performance evaluation of memory hierarchy for highly multithreaded graphics processors." ACM SIGPLAN Notices. Vol. 47.No. 8.ACM, 2012.

[10] Kim, Yooseong, and Aviral Shrivastava. "CuMAPz: a tool to analyze memory access patterns in CUDA." Proceedings of the 48th Design Automation Conference.ACM, 2011.

[11] Hong, Sunpyo, and Hyesoon Kim. "An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness."ACM SIGARCH Computer Architecture News.Vol. 37.No. 3.ACM, 2009.

[12] http://www.datasheetarchive.com/dl/Datasheets

[13] Che, Shuai, et al. "Rodinia: A benchmark suite for heterogeneous computing." Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on. IEEE, 2009.