Microarchitectural Characterization of Irregular Applications on GPGPUs

Tao Zhang
Department of Computer Science and Engineering
Shanghai Jiao Tong University, China 200240
tao.zhang@sjtu.edu.cn

Guangshuo Chen
Department of Computer Science and Engineering
Shanghai Jiao Tong University, China 200240
cgs.sjtu@gmail.com

Wei Shu
Department of Electrical and Computer Engineering
University of New Mexico, USA, NM 87131
shu@ece.unm.edu

Min-You Wu
Department of Computer Science and Engineering
Shanghai Jiao Tong University, China 200240
mwu@sjtu.edu.cn

ABSTRACT

In recent years, GPGPUs have experienced tremendous growth as general-purpose and high-throughput computing devices. However, irregular applications cannot fully utilize the hardware resource because of their plenty of control-flow divergences, irregular memory accesses and load imbalances. The lack of in-depth characterization and quantifying the ways in which irregular applications differ from regular ones on GPGPUs has prevented users from effectively making use of the hardware resource. We examine a suite of representative irregular applications on a cycle-accurate GPU simulator. We characterize their performance aspects and analyze the bottlenecks. We also assess the impact of changes in cache, DRAM and interconnect and discuss the implications for GPU architecture design. This work is useful in understanding and optimizing irregular applications on GPUs.

Keywords

Performance Characterization, Irregular, GPU, Microarchitectural

1. INTRODUCTION

GPUs are very efficient in accelerating regular applications [4]. However, many problem domains employ algorithms that are irregular in nature who exhibit input-dependent control flow and memory access patterns [2]. In this work, we characterize and analyze a suite of irregular and regular applications to show their differences and the performance bottlenecks. Burtscher et al. [2] studied a group of irregular applications on their control-flow and memory access irregularity on a GPU. Different with their work, we will utilize a cycle-accurate simulator to analyze more performance aspects. In addition, we'll assess the sensitivity of these applications to cache and DRAM latency and bandwidth, cache size, and coalescing behavior by modifying the simulator architectural parameters.

IFIP WG 7.3 Performance 2014, October 7-9, Turin, Italy. Copyright is held by author/owner(s).

Table 1: Applications and input characteristics (#K: number of kernels in applications' code)

Name	Abbr.	#K	Input
Barnes-Hut N-body Simulation [2]	BH	9	64K bodies, 1 time step
Breadth-First Search [2]	BFS	5	Com-Amazon (327K nodes, 904K edges); RMAT19 graph(512K nodes, 2048K edges);
Delaunary Mesh Refinement [2]	DMR	4	32K triangles; 64K triangles;
Minimum Spanning Tree [2]	MST	7	same as BFS
Points-to Analysis [2]	PTA	40	VIM 246K pointers, 108K constrains
Single-Source Shortest Paths [2]	SSSP	2	same as BFS
Survey Propagation [2]	SP	3	42K clauses, 10K literals, 3 literals per clause
Traveling Salesman Problem [2]	TSP	3	kroE100(100 cities, 200K climbers)
N-queens [1]	NQU	1	13 (the size of the problem)
Binomial Options [3]	BO	1	default
Coulombic Potential [1]	CP	1	2000 atoms on a 256×512 grid

2. EXPERIMENTAL SETUP

This section describes the applications, their inputs and the simulator used in the evaluation.

2.1 Applications

We characterize nine irregular applications including Traveling salesman problem [4], N-queens solver in the GPGPU-Sim sample applications [1], and all seven irregular applications in the latest LonestarGPU benchmark suit 2.0 [2]. In addition, we also evaluate two regular applications Binomial Options from CUDA SDK 4.2 [3] and Coulombic Potential from GPGPU-Sim sample applications [1] for comparison. The characteristics of the applications and their inputs are summarized in Table 1.

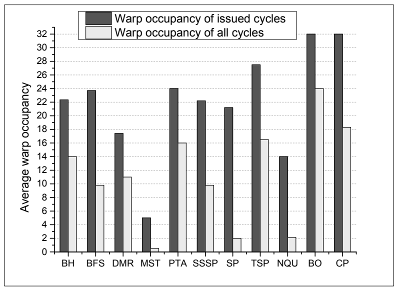

Figure 1: Average warp occupancy

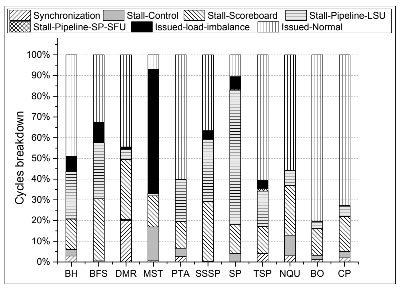

Figure 2: Cycles breakdown

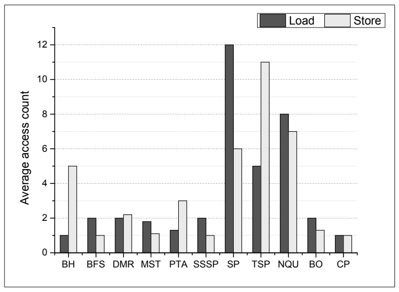

Figure 3: Average access count

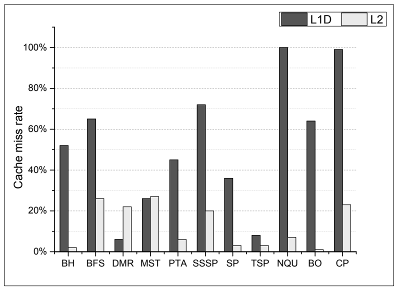

Figure 4: Cache miss rate

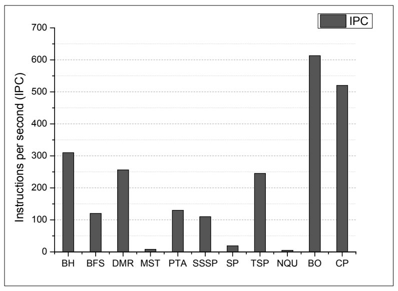

Figure 5: Instructions per cycle

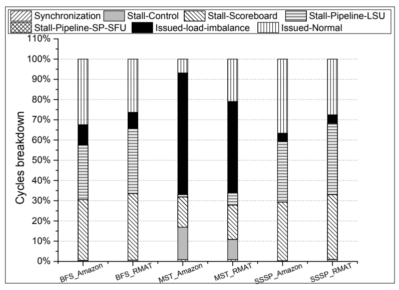

Figure 6: Input sensitivity

2.2 Simulator

The cycle-accurate GPGPU-Sim simulator [1] 3.2.1 and CUDA SDK 4.2 [3] (the highest version of CUDA that GPGPU-Sim supports) are adopted to execute the selected applications. An Nvidia GTX480 GPU (Fermi architecture) is simulated using the official configuration files in the simulator.

3. RESULTS

In this section, we first run the applications with the default GTX480 configurations and check their baseline performance details. Then we examine applications for dominant performance bottlenecks by varying several architectural components for memory accessing.

3.1 Characterizing baseline performance

Control-flow divergence is a common and major issue that degrades GPUs' performance. The average warp occupancy (number of active threads) can describe the intensity of control-flow divergences. Figure 1 plots the average warp occupancy of issued cycles and all cycles (issued cycles plus stall cycles), respectively. Compared to regular applications, irregular applications obtain a lower average warp occupancy in issued cycles. In addition, irregular applications with lots of stall cycles such as SP and MST have an even lower average warp occupancy for all cycles due to the lack of instruction-level parallelism or data-level parallelism.

Load imbalance is another problem that impairs system performance. Figure 2 plots the breakdown of applications' cycles. There is a special category "issued-load-imbalance" to identify the issued cycles where some warps have longer execution time than other warps within a same thread block. There are two observations. First, load imbalance only occur in irregular applications. Second, irregular applications have more stalls on the pipeline load & store units than regular applications since their irregular memory access patterns produce more un-coalesced memory accesses and pose pressure on limited number of load & store units.

Irregular, un-coalesced memory accesses underutilize the load & store units and interconnect bandwidth. Figure 3 plots the average number of memory accesses performed by each global or local load or store instruction. A value larger than one illustrates the presence of un-coalesced memory accesses. SP, TSP, NQU and BH exhibit larger average memory access count because of their irregular access patterns and the granularity to access memory.

Figure 4 presents the L1D and L2 cache miss rate of applications. Both regular and irregular applications have high cache miss rate, especially on the small L1D cache. The high miss rate is the result of un-coalesced memory accesses on GPUs and the memory access patterns of active threads who do not exploit spatial locality.

Figure 5 shows the overall performance in terms of instructions per cycle (IPC) for each application. We can see that regular applications BO and CP have higher IPC than irregular applications. The reason is that regular applications have less code branches and more coalesced memory accesses. Consequently, they achieve a higher utilization on the GPU computing resource.

Figure 6 presents the cycles breakdown for BFS, MST and SSSP with the first and the second input(see Table 1). There is a significant difference in the breakdown since the branches and memory accesses of these irregular applications are input-dependent.

3.2 Application bottleneck analysis

Figure 7 shows the applications' normalized performance

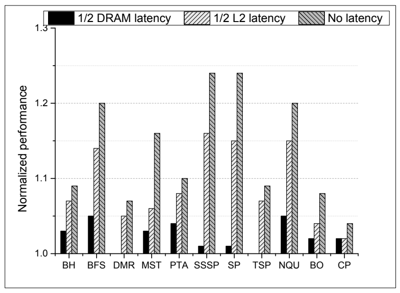

Figure 7: Impact of memory access latency on performance

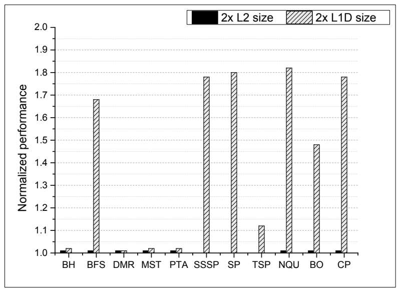

Figure 8: Impact of cache size on performance

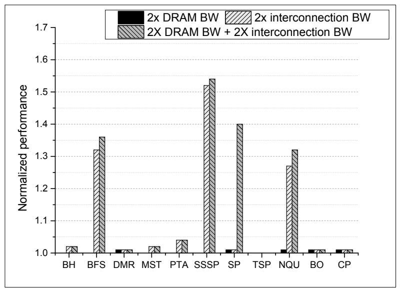

Figure 9: Impact of bandwidth on performance

under halved DRAM latency and L2 cache latency. The performance is normalized to that with un-modified simulator configurations. We can see that applications are more sensitive to the decreasing of L2 cache latency. Besides, irregular applications achieve larger performance improvements than regular applications. The Figure also presents the performance of an ideal system with ”no latency”.

Figure 8 presents the applications' normalized performance under double-sized L2 cache or L1D cache. On one hand, both regular and irregular applications can benefit from enlarged caches. On the other hand, their performance increases more significantly with enlarged L1D cache than with bigger L2 cache.

Figure 9 shows the applications' normalized performance when we change the bandwidth (BW) of the DRAM or the interconnection. The results show that applications are more sensitive to the increased bandwidth of the interconnect instead of the DRAM. A wider interconnect can carry memory access requests and replying data at a higher speed.

In summary, irregular applications are more sensitive to the changes of L2 latency and interconnection bandwidth than that of DRAM latency and bandwidth.

3.3 Summary analysis

In this subsection, we discuss each irregular application based on the results in previous two subsections.

BH: The tree-building kernel suffers from too many synchronization barriers, atomic operations, and imbalanced work due to different depth of the octree branches.

BFS: This kernel suffers from a huge number of LSU stalls due to its low compute to memory access ratio. In addition, its vertex-centric computing incurs significant branch divergences and load imbalances since vertices have different degrees.

DMR: There is a significant performance penalty in the refinement kernel since the handling of irregular graphs incurs a lot of memory access stalls and divergences.

MST: The dominant kernel adopts the vertex-centric model and each thread loops over its node's neighbors. As a result, there are lots of branch divergence, data hazards (scoreboard) and control hazards.

PTA: This kernel operates on a dynamically growing constraint-graph in which directed edges are added. Therefore, branch divergences and irregular memory accesses are unavoidable.

SSSP: This kernel has the same issue as BFS and MST.

SP: This kernel traverses a random input graph which incurs irregular and un-coalesced memory accesses. It further suffers from control hazard idle cycles due to the very short and irregular loop structures.

TSP: This kernel suffers from load imbalance and irregular memory accesses since threads within a warp take different cycles to find a local minimum. Fast threads/warps have to wait before they can move on to a new tour.

NQU: This kernel suffers from branch divergences, load imbalances and irregular memory accesses since each thread searches for a different sub-problem space for valid placements of N-queens.

4. CONCLUSION

The experiment results in this paper manifest that, compared to regular applications, irregular applications have lower average warp occupancy, more stalls at the scoreboard and the load & store units and exhibit load imbalances. Further experiments on tuning architectural components manifest that we can improve the performance of irregular applications more effectively by decreasing L2 access latency, enlarging L1D capacity and increasing the bandwidth of the interconnect.

Acknowledgment

The authors would like to thank anonymous reviewers for their fruitful feedback and comments. This research was supported by NSF of China under grant No. 61373155.

5. REFERENCES

[1] A. Bakhoda, G. L. Yuan, W. W. Fung, H. Wong, and T. M. Aamodt. Analyzing cuda workloads using a detailed gpu simulator. In *Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on*, pages 163–174. IEEE, 2009.

[2] M. Burtscher, R. Nasre, and K. Pingali. A quantitative study of irregular programs on gpus. In *Workload Characterization (IISWC), 2012 IEEE International Symposium on*, pages 141–151. IEEE, 2012.

[3] C. Nvidia. Sdk code samples.

[4] M. A. ONeil, D. Tamir, and M. Burtscher. A parallel gpu version of the traveling salesman problem. In *2011 International Conference on Parallel and Distributed Processing Techniques and Applications*, pages 348–353, 2011.