POSTER: Pagoda: A Runtime System to Maximize GPU Utilization in Data Parallel Tasks with Limited Parallelism

Tsung Tai Yeh, Amit Sabne, Putt Sakdhnagool, Rudolf Eigenmann, Timothy G. Rogers
Department of Electrical and Computer Engineering
Purdue University
{yeh14, asabne, psakdhna, eigenman, timrogers}@purdue.edu

ABSTRACT

Massively multithreaded GPUs achieve high throughput by running thousands of threads in parallel. To fully utilize the hardware, contemporary workloads spawn work to the GPU in bulk by launching large tasks, where each task is a kernel that contains thousands of threads that occupy the entire GPU.

GPUs face severe underutilization and their performance benefits vanish if the tasks are narrow, i.e., they contain less than 512 threads. Latency-sensitive applications in network, signal, and image processing that generate a large number of tasks with relatively small inputs are examples of such limited parallelism. Recognizing the issue, CUDA now allows 32 simultaneous tasks on GPUs; however, that still leaves significant room for underutilization.

This paper presents Pagoda, a runtime system that virtualizes GPU resources, using an OS-like daemon kernel called MasterKernel. Tasks are spawned from the CPU onto Pagoda as they become available, and are scheduled by the MasterKernel at the warp granularity. This level of control enables the GPU to keep scheduling and executing tasks as long as free warps are found, dramatically reducing underutilization. Experimental results on real hardware demonstrate that Pagoda achieves a geometric mean speedup of 2.44x over PThreads running on a 20-core CPU, 1.43x over CUDA-HyperQ, and 1.33x over GeMTC, the state-of-the-art runtime GPU task scheduling system.

CCS Concepts

•Software and its engineering → Runtime environments;

Keywords

GPU scheduling; many task computing

1. INTRODUCTION

GPGPU computing has demonstrated an ability to accelerate applications that have a high degree of parallelism, where iterations of large parallel loops are executed on the GPU. These applications see significant performance benefits since they can fully utilize the GPU's hardware resources by launching enough concurrent threads.

The GPU's performance benefits start to vanish as the degree of parallelism lessens. Conventionally, large parallel loops are offloaded to the GPU, while retaining the execution of smaller ones on the CPU. The main thesis of this paper is that despite having a smaller degree of parallelism, applications can benefit from using the GPU, provided that the involved task (or CUDA kernel) count is sufficiently high. Each such task, called a *narrow task*, has limited parallelism (< 512 data parallel threads in practice). Narrow tasks emerge in a number of scenarios. One set of such applications comprises latency-driven, real-time workloads. E.g. online sensors that generate small inputs, resulting in tasks with low parallelism; however, many such tasks are generated in quick succession and require immediate processing. These workloads have been characterized as having mixed task and data parallelism [4]. Secondly, irregular applications can exhibit narrow tasks. These applications often contain varying amounts of computation among different threads, and/or among loop iterations. To reduce load imbalance, these applications are often represented using many tasks with low degrees of parallelism [3]. Irregular workloads may also arise in multi-programmed environments. Different applications with low degrees of parallelism can be co-executed on a node to exploit all the computing resources. Each of these applications may launch one or more tasks of different computation sizes, resulting in narrow tasks.

GPU underutilization is the key reason why narrow tasks are conventionally executed on the CPU. This paper presents Pagoda, a runtime system that highly improves GPU utilization in the presence of narrow tasks. Pagoda introduces novel elements of a massively parallel OS to flexibly and efficiently virtualize and dynamically schedule GPU core resources at the warp granularity, enabling hundreds of tasks to execute concurrently.

2. PAGODA RUNTIME SYSTEM DESIGN

Pagoda is a CPU/GPU system that overrides the traditional CUDA kernel launching API. Pagoda's CPU component handles the spawning and mapping of tasks to different streaming multiprocessors (SMX), while a dameon-like MasterKernel is launched to override CUDA's default threadblock scheduler with Pagoda's software scheduler. Pagoda tracks the many tasks it is capable of running in a novel data structure called the TaskTable (shown in Figure 2),

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
PACT '16 September 11-15, 2016, Haifa, Israel
© 2016 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-4121-9/16/09.
DOI: http://dx.doi.org/10.1145/2967938.2974055

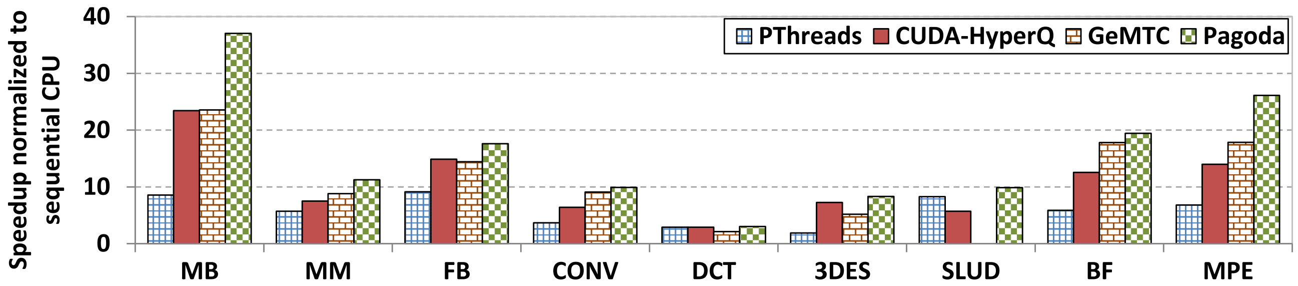

*Figure 1: **Overall Performance Comparison:** The number of tasks in each benchmark is constant (32K), except SLUD, which contains 273K tasks. Each GPU task uses 128 threads. The measurement of execution time contains both data copy and compute times. Pagoda significantly outperforms CUDA-HyperQ(1.43x), 20-core PThreads(2.44x), and GeMTC(1.33x), owing to the higher utilization. GeMTC sees high overheads when task execution times are low, such as in DCT and 3DES.*

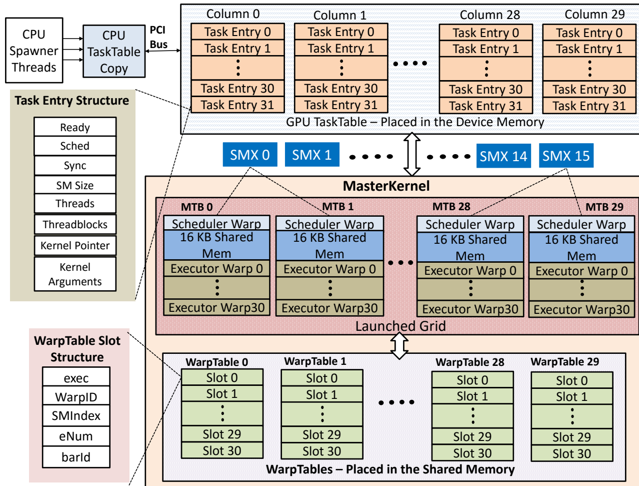

*Figure 2: **The MasterKernel Composition in the NVIDIA Tesla K40 GPU:** The 30 MasterKernel threadblocks (MTBs) have 1024 threads and 16 KB shared memory each. TaskTable is mirrored on both the CPU and GPU. Each entry of the task and warp table contains multiple flags to record states of scheduling. Two columns of warp table correspond to one SMX.*

which is stored in both CPU and GPU memory. The distributed Task:Table features a lazy copy mechanism that enables efficient task launching and scheduling. Since Pagoda overrides CUDA's default threadblock scheduler, it supports the native CUDA runtime model by implementing an efficient shared memory allocator in software and by managing named barriers to emulate the behaviour of __syncthreads.

3. EVALUATION

Figure 1 plots the performance of Pagoda and several other solutions. The GPU implemenations use an NVIDIA Tesla K40, while the PThreads and sequential implementations use two hyper-threaded 10 core Intel Xeon CPUs. Figure 1 shows that Pagoda obtains higher performance than other runtime schemes, namely, PThreads on the CPU, and CUDA-HyperQ [2] and GeMTC [1] on the GPU. The speedup is calculated over the entire execution time, i.e, it measures both compute and data copy times. The benefit is attributed to the high utilization achieved by Pagoda. The GeMTC versions do not use shared memory, since GeMTC has no support for it. We could not implement SLUD in GeMTC; GeMTC needs the number of tasks to be predefined, which is not the case in SLUD. SLUD represents an interesting case where HyperQ is slower than PThreads, but Pagoda is faster. Speedup achieved by all GPU schemes is low in DCT since the application is data copy bound. We tried OpenMP data parallelism, OS-based task scheduling, Python-based thread pooling, and PThreads-based task-level parallelism on the CPU to run narrow tasks. PThreads obtained the best results, which we include in Figure 1.

4. CONCLUSION

Pagoda is a GPU runtime system that overcomes underutilization in the presence of narrow tasks. Pagoda virtualizes GPU resources via MasterKernel, a persistently executing daemon on the GPU. Pagoda launches tasks on the GPU as long as some free warps are available. Unlike previous work, Pagoda supports all functionality of the native CUDA model. A key distinction in Pagoda is the task spawning and scheduling mechanism. Pagoda's novel TaskTable, which is shared between the CPU and GPU reduces handshaking during the spawning process to greatly reduce overhead.Pagoda achieves concurrent task scheduling, and overlaps task spawning, scheduling, and execution through pipelining. The experimental evaluation showed that Pagoda achieves geometric mean speedups of 1.43x over CUDA-HyperQ, 1.33x over GeMTC, and 2.44x over 20-core CPU Pthreads. The evaluation also showed that Pagoda can outperform static fusion schemes by 2.31x, and achieves much lower latency per task.

5. REFERENCES

[1] S. J. Krieder, J. M. Wozniak, T. Armstrong, M. Wilde, D. S. Katz, B. Grimmer, I. T. Foster, and I. Raicu. Design and evaluation of the gemtc framework for gpu-enabled many-task computing. In *Proceedings of the 23rd International Symposium on High-performance Parallel and Distributed Computing*, HPDC '14, pages 153–164, New York, NY, USA, 2014. ACM.

[2] NVIDIA. Hyper-Q Example. [Online]. Available: http://docs.nvidia.com/cuda/samples/6_Advanced/simpleHyperQ/doc/HyperQ.pdf, 2012. (accessed March. 5, 2016).

[3] K. Ousterhout, A. Panda, J. Rosen, S. Venkataraman, R. Xin, S. Ratnasamy, S. Shenker, and I. Stoica. The case for tiny tasks in compute clusters. In *Presented as part of the 14th Workshop on Hot Topics in Operating Systems*, Berkeley, CA, 2013. USENIX.

[4] J. Subhlok and G. Vondran. Optimal use of mixed task and data parallelism for pipelined computations. *J. Parallel Distrib. Comput.*, 60(3):297–319, 2000.