POSTER: Performance Modeling for GPUs using Abstract Kernel Emulation

Changwan Hong[1] Aravind Sukumaran-Rajam[1] Jinsung Kim[1] Prashant Singh Rawat[1]
Sriram Krishnamoorthy[2] Louis-Noël Pouchet[3] Fabrice Rastello[4] P. Sadayappan[1]
[1] Ohio State University, {hong.589,sukumaranrajam.1,kim.4232,rawat.15,sadayappan.1}@osu.edu
[2] Pacific Northwest National Laboratory, sriram@pnnl.gov
[3] Colorado State University, pouchet@colostate.edu
[4] INRIA, fabrice.rastello@inria.fr

Abstract

Performance modeling of GPU kernels is a significant challenge. In this paper, we develop a novel approach to performance modeling for GPUs through abstract kernel emulation along with latency/gap modeling of resources. Experimental results on all benchmarks from the Rodinia suite demonstrate good accuracy in predicting execution time on multiple GPU platforms.

CCS Concepts • **Computing methodologies → Modeling methodologies**;

1 Introduction

Optimizing compilers generally use highly simplified performance models due to the significant challenges in developing accurate analytical performance models for complex computer systems. Although several research efforts [1, 5] have addressed analytical performance modeling for GPUs, none of them has demonstrated good accuracy over a wide range of GPU kernels. In this paper, we pursue a different approach than prior efforts towards performance modeling. Instead of an analytical modeling approach, we perform abstract emulatation of the kernel's binary code (SASS code generated by Nvidia's NVCC compiler) to track the start and completion times for each instruction from a maximal set of thread-blocks that can be concurrently loaded on one Streaming Multiprocessor (SM). Key system resources—global-memory subsystem, shared-memory, and functional-units—are modeled using two *parameters*: 1) *latency*, the number of clock cycles for a request, from entry to exit; 2) *gap* (inverse throughput), the minimum number of cycles between successive request completions at that resource. The effectiveness of the proposed approach is demonstrated via experimental results on two GPU platforms, using all kernels in the Rodinia [3] benchmark suite.

2 Abstract Kernel Emulation

This section details the algorithm for abstract kernel emulation. The algorithm uses parameters (*latency* and *gap* for each resource) that characterize the target GPU architecture. The parameters are obtained using micro-benchmarks, similar to those described by Papadopoulou et al. [2]. An example to illustrate abstract kernel emulation is provided in Fig. 1, considering an SM with the following resource parameters: global-memory latency = 500; global memory gap = 100; functional unit latency = 100; functional unit gap = 20.

The figure shows different stages of abstract kernel emulation with 3 warps. Initially the first statement from the first

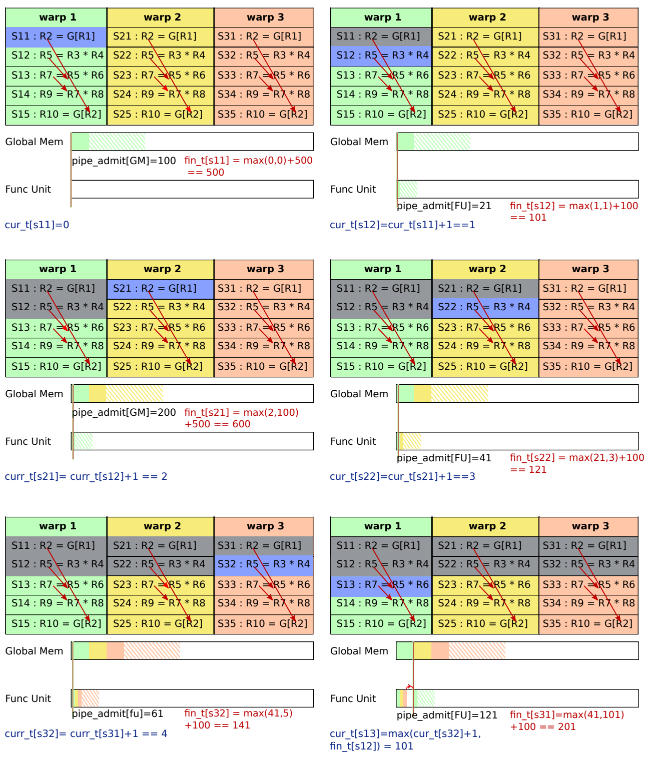

Figure 1. Illustration of Abstract Kernel Emulation

PPoPP '18, February 24–28, 2018, Vienna, Austria
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-4982-6/18/02...$15.00
https://doi.org/10.1145/3178487.3178524

warp is selected. Statement $S11$ reads a value from global-memory and assigns it to a register. Since $S11$ is not dependent on any other instruction and the global-memory is ready to accept/admit a request, the statement can be issued immediately. Once $S11$ is issued, global-memory requests are blocked for 100 cycles (the *gap* parameter of the global-memory resource. $S11$ will require 500 cycles (global-memory *latency*) to finish its execution. The next instruction, $S12$, is an add instruction and thus requires the functional unit. Since $S12$ does not depend on any previous instruction and the functional unit is ready for requests, it can be scheduled at clock cycle 1. $S12$ will require 100 cycles to finish execution. The next instruction of warp 1, $S13$, is dependent on $S12$. Since $S12$ will only complete at clock cycle 101, $S13$ cannot be scheduled immediately and we move to warp 2. $S21, S22, S31, S32$ follow the same pattern as $S11$ and $S12$. $S32$ is scheduled in clock cycle 4 and completes execution in clock cycle 141. At clock cycle 5, no warp has an instruction that can be scheduled. The next instruction that can be scheduled is $S13$ at clock cycle 101. The clock is advanced to step 101 and $S13$ is selected for scheduling.

The abstract kernel emulation begins by scheduling the first instruction from warp 0 and proceeds until all instructions from all warps are scheduled. During abstract kernel emulation, ready instructions from the active warps are scheduled for execution using a Greedy Then Oldest (GTO) [4] scheduling. The completion timestamp of an instruction is computed by adding the latency of the various resources it goes through (e.g., shared-memory, global-memory, special functional units, etc.). For each resource, a *pipe_admit time* is maintained, which is incremented by its gap when a new request starts execution on it. Dependences between instructions are tracked, so that the earliest schedulable time for an instruction is made later than the completion times of all previously scheduled instructions it depends on. After all the instructions are scheduled, the clock is set to the latest finish time of the last completing instruction among all warps. Several details such as conditional statements, irregular/uncoalesced data access, caches, barrier synchronization, and bank conflicts are also modeled to improve accuracy.

3 Experiments

We present an experimental evaluation of prediction accuracy of abstract kernel emulation on two GPU systems: Nvidia K20c with 13 Kepler SMs, 5GB global memory, 706MHz, 1.25MB L2 cache, and 48KB shared memory, and Nvidia Titan X with 28 Pascal SMs, 12GB global memory, 1417MHz, 4MB L2 cache, and 96KB shared memory. We evaluate all the 58 kernels in the Rodinia benchmark suite. Those kernels collectively include 369 if-then-else conditions and loop bounds dependent on known parameters, 155 if-then-else conditions dependent on dynamically computed values, and 11 loops with statically unresolvable loop bounds.

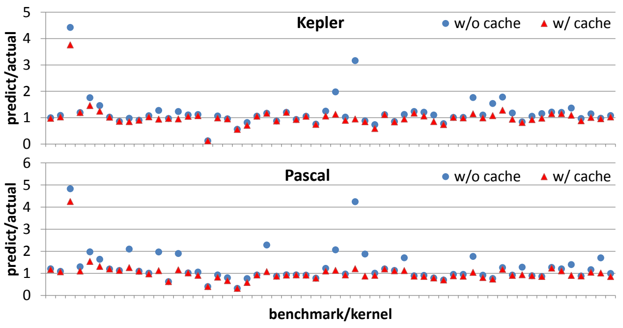

Figure 2. Modeling accuracy with Rodinia benchmarks

Each kernel's binary was extracted and the SASS code was subjected to abstract kernel emulation, using the respective parameters for both targeted GPUs. The benchmarks were also executed to measure actual execution time. For each kernel in each benchmark, the ratio of predicted execution time to actual execution time on the Kepler and Pascal system are shown in Fig. 2. The plots show two sets of data, respectively with and without including cache effect modeling. It may be seen that the impact of including cache modeling is quite significant for some of the kernels. Cache miss rate parameters can be obtained by incorporating a cache miss prediction model, or, as done here, actual measurement (NVPROF is used). For both machines, the execution time is predicted quite well for a majority of the kernels, especially when cache modeling is included.

4 Conclusion

We have developed an approach to modeling performance of GPU kernels via abstract kernel emulation of a small number of thread blocks for kernels and modeling each hardware resource using latency and gap parameters. Experimental evaluation using all kernels of the Rodinia benchmark suite demonstrate good accuracy. The modeling approach is being used in ongoing work for manual as well as auto-tuned performance optimization of GPU kernels.

Acknowledgments

This work was supported in part by the U.S. National Science Foundation via awards 1440749 and 1513120.

References

[1] Sara S Baghsorkhi, Matthieu Delahaye, Sanjay J Patel, William D Gropp, and Wen-mei W Hwu. 2010. An adaptive performance modeling tool for GPU architectures. In *PPOPP '10*.
[2] Misel-Myrto Papadopoulou, Maryam Sadooghi-Alvandi, and Henry Wong. 2009. Micro-benchmarking the GT200 GPU. *Computer Group, ECE, University of Toronto, Tech. Rep* (2009).
[3] Rodinia. 2009. Accelerating Compute-Intensive Applications with Accelerators. (2009). https://www.cs.virginia.edu/~skadron/wiki/rodinia
[4] Timothy G Rogers, Mike O'Connor, and Tor M Aamodt. 2012. Cache-conscious wavefront scheduling. In *MICRO '12*.
[5] Jaewoong Sim, Aniruddha Dasgupta, Hyesoon Kim, and Richard Vuduc. 2012. A performance analysis framework for identifying potential benefits in GPGPU applications. In *PPOPP '12*.