PPT-GPU: Scalable GPU Performance Modeling

Yehia Arafa[1], Abdel-Hameed A. Badawy[1,2], Gopinath Chennupati[2], Nandakishore Santhi[2], Stephan Eidenbenz[2]
[1] Klipsch School of ECE, New Mexico State University. Las Cruces, NM 88003, USA
[2] Los Alamos National Laboratory. SM 30, Los Alamos, NM 87545, USA
{yarafa, badawy}@nmsu.edu {gchennupati, nsanthi, eidenben}@lanl.gov

Abstract—Performance modeling is a challenging problem due to the complexities of hardware architectures. In this paper, we present PPT-GPU, a scalable and accurate simulation framework that enables GPU code developers and architects to predict the performance of applications in a fast, and accurate manner on different GPU architectures. PPT-GPU is part of the open source project, Performance Prediction Toolkit (PPT) developed at the Los Alamos National Laboratory. We extend the old GPU model in PPT that predict the runtimes of computational physics codes to offer better prediction accuracy, for which, we add models for different memory hierarchies found in GPUs and latencies for different instructions. To further show the utility of PPT-GPU, we compare our model against real GPU device(s) and the widely used cycle-accurate simulator, GPGPU-Sim using different workloads from RODINIA and Parboil benchmarks. The results indicate that the predicted performance of PPT-GPU is within a 10% error compared to the real device(s). In addition, PPT-GPU is highly scalable, where it is up to 450x faster than GPGPU-Sim with more accurate results.

Index Terms—GPGPU, performance prediction, software/hardware co-design, GPU modeling, PPT

1 INTRODUCTION

Graphics processor units (GPUs) originally designed to render images to display(s) have evolved to be powerful co-processors that perform complex calculations efficiently. This drives the need for fast and accurate ways to evaluate the performance of different applications under various GPU architectures. However, due to the complexities found in these hardware architectures, modeling and predicting the runtime of GPUs have become a non-trivial challenge and an active area of research for many years.

Over the years, researchers have proposed many analytical approaches [1], [2], [3] to model the runtime of GPUs. Analytical methods have the advantage of being fast but usually lack the accuracy required to capture all the dynamic effects of different kernels. On the other hand, cycle-accurate simulators have been widely accepted and used by the research community due to the acceptance of cycle-accurate modeling paradigm and the relative ease of use. For instance, GPGPU-Sim [4], is the standard cycle-accurate simulator used in the GPGPU architecture research. Cycle-accurate simulators usually have more accurate results than analytical models but at the cost of simulation speed. Moreover, the size of the datasets is limited.

In addressing these challenges, we introduce a scalable and accurate performance prediction framework on GPUs, PPT-GPU. PPT-GPU is a hybrid model between analytical and cycle-accurate approaches that can predict the runtime behavior of GPU workloads under different GPU architectures. PPT-GPU can run much larger datasets without sacrificing the accuracy, the scalability, and the speed of obtaining results. At the source code level, we divide the total computational workload into small parallel sub-workloads that run concurrently on the available SMs and estimate the runtime from the instruction mix of the sub-workload occupying one SM to predict the full workload performance.

PPT-GPU is part of the open source project, Performance Prediction Toolkit (PPT) [5], a scalable co-design framework developed at the Los Alamos National Laboratory. PPT has parameterized hardware and middleware models, accepts source code as the input, and predicts runtimes. PPT relies on Simian [6], a parallel discrete event simulation engine written in Lua and Python.

Manuscript submitted: 08-Nov-2018. Manuscript accepted: 28-Jan-2019. Final manuscript received: 08-Mar-2019

The old GPU model [7] in PPT predicts the runtime of applications while ignoring various components in a modern GPUs, as a result, the runtime predictions are inaccurate. The aim of this work is to improve the old GPU model by considering the following: (1) Different memory hierarchies of a modern GPU; (2) The latencies of instruction(s) of a GPU, these latencies are dependent on the instruction itself and the GPU family; (3) Validating PPT-GPU against real GPU devices and GPGPU-Sim [4] on a set of 10 kernels from both RODINIA [8] and Parboil [9] benchmarks. Our results suggest that the new PPT-GPU significantly improves the prediction accuracy while maintaining a scalable performance prediction.

Our model assumes that global (and local if there are any) memory instructions bypass the L1 and L2 caches to fetch data from the GPU device memory. But given that caches can have a high impact on the application performance, we also show the effects of adding the L2 cache to the prediction accuracy of PPT-GPU. Building a model that can predict the performance of L1/L2 caches is beyond the scope of this paper and is part of our ongoing work. We utilize an oracle using NVIDIA Visual Profiler tool [10] to predict the L2 cache hit rate of applications, then we feed that into PPT-GPU as input and this gives us an upper bound to the effect of L2 caches on our prediction results.

The rest of the article is organized as follows: Section 2 explains the architecture of NVIDIA GPUs and CUDA terminology; Section 3 shows the PPT-GPU architecture; while Section 4 shows our prediction results; Section 5 discusses relevant related work; finally, Section 6 concludes the paper and presents future work.

2 GPU ARCHITECTURE

A typical GPU consists of a number of streaming multiprocessors (SM), each can be seen as a standalone processor. Specifically, each SM has its own instruction buffer, scheduler, and dispatch unit. In addition, each has a number of computing resources that handle different ALU operations. All SMs share access to a shared off-chip global memory. In GPUs global memory is the largest, and most commonly used. However, accessing the global memory frequently hurts the performance due to the fact that global memory access latency is high compared to other on-chip memories. As a result, architects

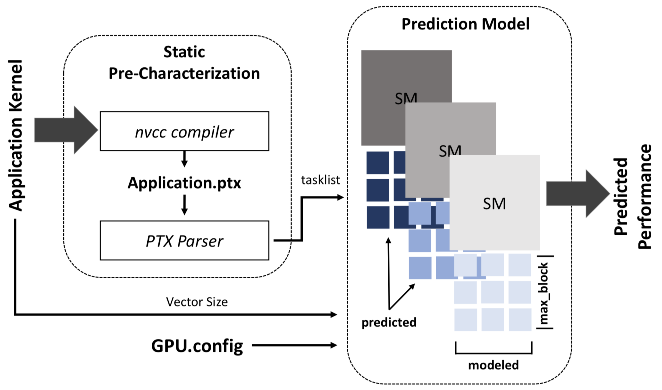

Fig. 1. PPT-GPU Architecture Overview. PPT-GPU is fully parameterized software that takes the configurations of the GPU to be modeled as input along with the vector size, that is the size of the workload in terms of the total number of grids and blocks of the application.

introduced on-chip L1 & L2 cache memories to GPUs, with goals of reducing the traffic to the global memory. There is one L1 cache per SM and one L2 cache shared among all SMs. Furthermore, each SM has a small amount of shared memory used for inter-thread communication.

In CUDA terminology all threads spawned from the same kernel are grouped into a grid, all of which share the same global memory. The grid is made up of many thread blocks each of which is composed of groups of 32 threads called warps. Grids and blocks represent a logical view of the thread hierarchy of a CUDA kernel. In order to have high parallelism, warps execute instructions in a SIMD manner, meaning that all threads from the same warp execute the same instruction at any given time. Each SM contains a number of warp schedulers, which try to issue one or more instructions to a given warp at every clock cycle where instructions can be issued before the previous one if there is no dependency between them. CUDA gives the programmer additional avenue of optimization by exposing the abstraction of choosing the number of grids and blocks for an application which can have a high effect on the overall performance of the application.

3 PPT-GPU

We divide the runtime prediction into two phases as shown in figure 1: (i) Static Pre-Characterization, where PPT-GPU interacts with the application's source code to get the architecture independent tasklist. (ii) Prediction Model, a parameterized GPU model that predicts the runtime using the tasklist.

3.1 STATIC PRE-CHARACTERIZATION

The pre-characterization phase is done statically without the need to run the application binaries on a real GPU device. First, the application's intermediate representation, parallel thread execution (PTX) [11] serves as input to our framework. Since PTX is a machine-independent ISA, converting the application to its corresponding PTX needs to be done only once for various GPU architectures. Then this PTX file is parsed automatically to extract a tasklist for each kernel in the application. The tasklist is the way PPT-GPU interacts with the application since it represents the sequence of instruction for the code to be simulated. We choose this simple representation as with it's machine-in-dependency it can help architects simulate this code on future architectures. Table 1 shows a one-to-one mapping example of PTX instructions to our tasklist. An item in the tasklist represents an instruction which can be an ALU operation or a memory operation. The ALU instruction such as *add*, *sub*, *mul*, *etc.* is represented by *iALU* or *fALU* depending on whether it is an integer or a floating point instruction. This item will be issued to one of the SP units later unless it is double precision, it will then be issued to one of the DP units. If the instruction is a special function instruction like *cosine*, *sqrt*, *etc.* it is donated by *SFU* in the tasklist. The memory instruction, on the other hand, is represented by the type of the accessed memory space (global, local, texture, shared or constant) along with the type of access, read or write. If an instruction depends on another instruction the number of the latter is written in the former's tasklist item. If two (or more) consecutive instructions are independent, the second instruction can be issued before the completion of the first one. For example, the third instruction. in table 1 depends on the first and the second instructions., therefore it cannot be issued before them. However, the second instruction can be issued before the first one as there is no dependency. Instructions are issued at the warp granularity depending on the issuing capability of the GPU architecture modeled. Thus, multiple instructions can be issued at the same time if there is are no dependencies between them.

TABLE 1
PTX to tasklist conversion. This simple program loads two numbers from the GPU device memory (global memory space), adds them together, multiply the output by 5, and then stores it back to the device memory. Each PTX instruction has its corresponding one in the tasklist.

PTX	tasklist
ld.global.s32 %r1, [%rd4]	[GLOB_MEM_ACCESS, LOAD]
ld.global.s32 %r2, [%rd3]	[GLOB_MEM_ACCESS, LOAD]
add.s32 %r3, %r2, %r1	[iALU, **6**, 1, 2]
mul.lo.s32 %r5, %r3, 5	[iALU, **13**, 3]
st.global.s32 [%rd4], %r5	[GLOB_MEM_ACCESS, STORE, 4]

Since different ALU instructions require different cycles to finish execution instruction latencies are added to the items in the tasklist while parsing the PTX file depending on the type of the instruction and the GPU architecture. All the instructions found in PTX ISA are modeled with a corresponding latency. Table 2 shows an example of different instructions latencies as reported by Andersch *et al.* [12]. These latencies are used later for the number of cycles each instruction spends in the compute resource. For instance, in the program found in table 1, if it is to run on a GPU from Maxwell architecture, the first ALU operation is an integer add instruction that needs 6 cycles to finish execution while the second ALU operation is an integer multiply instruction that needs 13 cycles to finish execution.

3.2 PREDICTION MODEL

Given the grid and the block sizes of a kernel as a function of the input vector size, PPT-GPU computes the maximum number of active blocks that can run concurrently on a single SM (*max_block*) and the total number of warps that can be allocated per block statically. Depending on the GPU architecture, CUDA limits the number of blocks and warps that can run concurrently on a single SM but such limits are idealistic. The actual number of blocks and warps executing on a single SM will depend on the launch configuration and the resources available for that kernel (shared memory size, number of registers, number of threads, *etc.*). The total workload for a given kernel is divided into parallel sub-workloads depending on the grid size and the *max_block*. Since, PPT-GPU is a hybrid approach between analytical models and cycle-accurate models. We estimate the execution time the modeled GPU takes to

TABLE 2
Different instructions & memory latencies for Maxwell GPU architecture; {i} & {f} stands for integer & floating point respectively

Instruction	Latency	Memory unit	Latency
{i/f}ADD, {i/f}SUB	6	Shared Memory	28
(iMUL, fMUL)	(13, 6)	Texture Memory	330
(iDIV, fDIV)	(210, 374)	Constant Memory	184
(AND, OR, XOR)	6	Global Memory	350
(iMAD, fMAD)	(13, 6)	L1 Cache	-
(SIN, SQR)	(15, 128)	L2 Cache	194

execute a sub-workload using its instruction mix statically as mentioned in Section 1. We assume that all sub-workloads are divided evenly on the available SMs and processed in parallel. If any sub-workload(s) are unprocessed, we schedule it to one of the available SMs and update the runtime. To measure the runtime of the sub-workload, instructions from the tasklist are issued to a given warp depending on the number of warp schedulers available on the modeled GPU at each cycle. A warp can only issue a new instruction from the tasklist, if there are resources available to process the instruction without any dependencies. Then, we process warp instructions using parallelism and time slicing techniques. Depending on the instruction type, a compute resource will be requested, each compute resource can process a certain number of instructions in parallel depending on the modeled GPU configuration. If a resource cannot accept any new instructions, this instruction will wait until one of the compute resources can accept new instructions. The instruction will then spend the given latency occupying that resource.

Memory operations are also issued per warp. Table 2 shows the latencies of various memories units found in a modern GPU. For simplicity, we assume that all memory accesses for all the threads in a given warp are sequential and aligned, and therefore coalesced. We ignore memory divergence. In addition, we assume load and store accesses bypass L1 and L2 while accessing the device memory, which gives a worst-case prediction. Then, we calculate the memory access time for each memory instruction in the pipeline which depends on multiple factors taken from the configuration file, including the number of memory ports, memory bandwidth, and the size of memory concurrent transfer from the memory queue. Since cache can have a high impact on the application performance, we use an oracle to predict the L2 hit rate and feed it to our model. We then use this hit rate to compute a standard average memory access time (*AMAT*) [13] which is used for all instruction accessing the device memory space that is global or local memory instructions.

PPT-GPU does not take into account branch divergence, therefore we direct the nvcc compiler to optimize short conditional segments of the kernel by inserting predicated instructions. This takes care of branch divergence for short segments. For long conditional segments that cannot be predicated, a worst case scenario is assumed, where the corresponding instructions of the two paths are parsed and added to the tasklist to be executed by different warps in the model.

4 PREDICTION RESULTS

4.1 EXPERIMENTAL SETUP

We used GeForce GTX TITAN X & Tesla K40m graphics cards in our evaluation. We collected the specifications from non-academic sources such as graphics card databases and online reviews. Table 3 shows the specifications of the two used cards.

TABLE 3
Target GPUs Specifications

Configuration	TITAN X	K40m
Top Level configuration		
Architecture	Maxwell 2.0	Kepler
compute capability	5.2	3.5
Clock Domain (core/shader)	1000 MHZ	745 MHZ
Memory clock	1753 MHZ	1502 MHZ
Number of {Cores, SMs}	{3072, 24}	{2880, 15}
SM Level		
{SP, SFU} units per SM	{128,32}	{192,32}
Load/Store units per SM	32	32
Registers per SM	65536	65536
{Blocks, warps} per SM	{32, 64}	{32, 64}
Warp schedulers	4	4

TABLE 4
Experimental Benchmarks; for each workload, (P) & (R) stand for Parboil and RODINIA Benchmark suites, respectively

Workload Name (Suite)	Total Num of (Grids, Blocks)	Predicted/SM (Blocks, Warps)
SGEMM (P)	(528, 128)	(16, 64)
STENCIL (P)	(1024, 128)	(16, 64)
TPCAF (P)	(201, 256)	(8, 64)
HOTSPOT (R)	(7396, 256)	(6, 48)
LUD_3 (R)	(16129, 256)	(8, 64)
NN (R)	(168, 256)	(7, 56)
PATHFINDER (R)	(463, 256)	(8, 64)
GAUSSIAN (R)	(65536, 32)	(32, 32)
BFS_1 (R)	(1954, 512)	(4, 64)
BFS_2 (R)	(1954, 512)	(4, 64)

To have a meaningful performance comparison and to be fair in comparing PPT-GPU against other simulation frameworks, we have added a TITAN X configuration in GPGPU-Sim [4], we use GPGPU-Sim stable version 3.2.2. Since this GPGPU-Sim version only models only the Fermi GPU family. Hall [14] studied how to adapt GPGPU-Sim to new GPU architectures and used similar configurations to what we used for TITAN X GPU.

4.2 RESULTS

We evaluate our framework on different workloads from RODINIA [8] and Parboil [9] benchmark suites. Table 4 lists the benchmarks that used. We configured each benchmark's kernel to run only once and if a benchmark had more than one kernel we report the result of each kernel separately. We used the default datasets that came with the benchmarks and if there are multiple sizes we always use the biggest.

We report the performance in terms of the number of instructions per cycle (IPC). In order to validate PPT-GPU against a real device, we used the NVIDIA Visual Profiler tool [10]. We run the application ten times and took the median of the runs. Since we can enable or disable caches on the real device at compile time, we report the performance of the real device(s) once while disabling the L1 cache and using the L2 cache only and once while using both caches.

Figure 2 shows the prediction results for a TITAN X GPU. The prediction results are normalized to a real device while enabling only the L2 cache. The L1 cache effect is fairly small and highly application dependable as shown in Figure 2. The

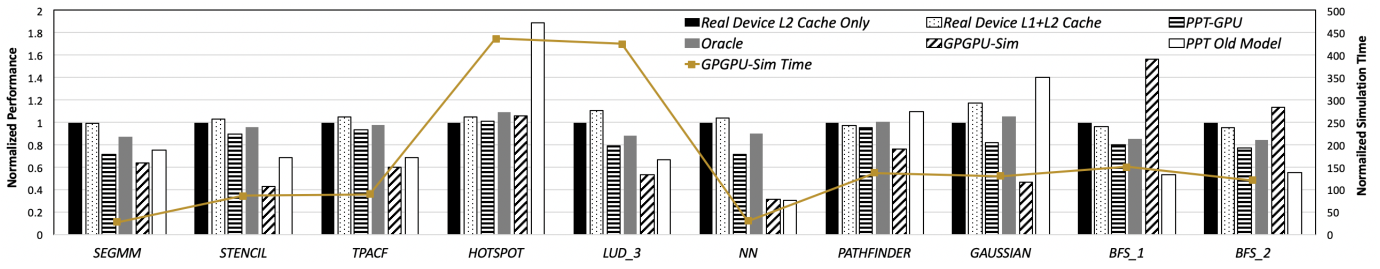

Fig. 2. GTX TITAN X prediction results. PPT-GPU is compared to real device, GPGPU-Sim, and the old model included in PPT. All the results are normalized to the real device while enabling L2 cache only on the left hand axis. Oracle is our prediction results after taking into account the optimistic L2 hit rates. The right hand axis shows the normalized simulation time of GPGPU-Sim relative to PPT-GPU.

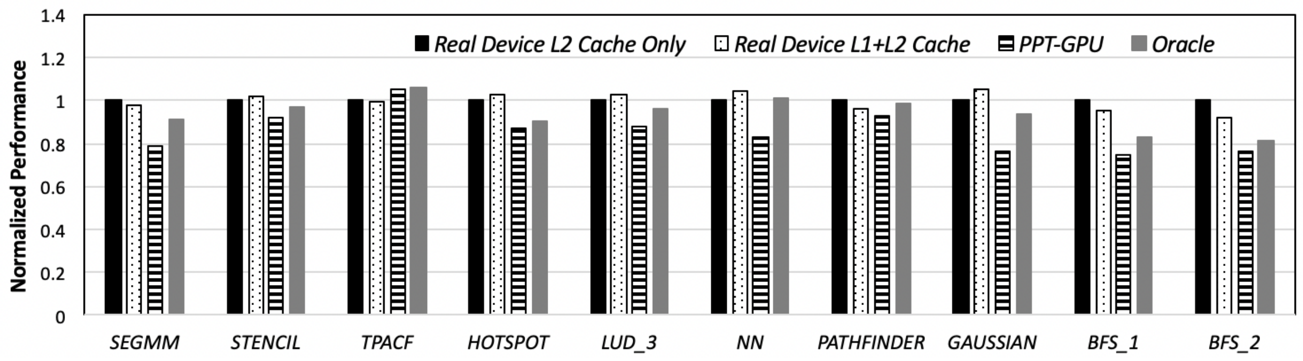

Fig. 3. Tesla K40m GPU prediction results. All the results are normalized to the real device while enabling L2 cache only. Oracle is our prediction results after taking into account the optimistic L2 hit rates. The figure shows the same set of benchmarks in the same order in the graph from Figure 2.

results show that the new PPT-GPU model performs better compared to GPGPU-Sim and the old circa 2015 model [7]. The results indicate that modeling the memory system and taking into account the instruction latencies significantly improve the performance predictions, which closely follow the real device. For instance, the old model was over predicting in HOTSPOT, PATHFINDER, and GAUSSIAN due to the absence of modeling the latencies for the different instructions executing in the GPU pipelines.

PPT-GPU results were further improved after adding the L2 hit rates taken from the Oracle. On average PPT-GPU predictions are within 10% of the real device performance. In addition, we have better accuracy than GPGPU-Sim. Furthermore, PPT-GPU is more scalable than GPGPU-Sim. We compare the simulation runtimes for both PPT-GPU and GPGPU-Sim in Figure 2 which shows as the solid line that corresponds to the left y-axis. We normalize PPT-GPU simulation time to the simulation time of GPGPU-Sim. For some benchmarks such as SGEMM and NN, PPT-GPU is **29x** faster than GPGPU-Sim, while for others such as HOTSPOT and LUD, PPT-GPU is **450x** faster. On average across all 10 benchmarks, PPT-GPU is **160x** faster than GPGPU-Sim. These results indicate that PPT-GPU is highly scalable.

Figure 3 shows the results for Tesla K40m GPU. Since Figure 2 proved that PPT-GPU is more accurate than GPGPU-Sim, we only present the results with respect to the real device. The results show that the global trends of the predicted performance for both modeled GPUs are within the same range.

5 RELATED WORK

Hybrid models have been proposed before. Punniyamurthy *et al.* [15] proposed a GPU abstract timing simulator. They have accurate predictions but their model relies on GPGPU-Sim. In addition, they run the application through a real GPU device first to capture dynamic microarchitecture effects. On the other hand, PPT-GPU is a standalone framework that statically analyzes the application without running it. Konstantinidis *et al.* [16] proposed an automated GPU performance prediction technique based on the roofline modeling [17] but their average error rate is 27% with some applications having more than 50% error rate. Lee *et al.* [18] proposed a prototype system for performance modeling from source code, COMPASS. It relies on Aspen [19], where the parameterized performance models are produced with a static analysis of the source code. Although, both COMPASS and PPT-GPU use static analysis, PPT-GPU models different components of the GPU architecture without sacrificing the ease of use, accuracy and scalability.

6 CONCLUSION & FUTURE WORK

In this paper, we present PPT-GPU, a fast and scalable performance prediction framework for GPUs. PPT-GPU is a fully parametrized tool that relies on (PTX) ISA and GPU configurations to predict applications' runtime without having to run them. PPT-GPU is part of the recently open sourced Performance Prediction Toolkit (PPT) [5]. We validated PPT-GPU with different workloads from RODINIA and Parboil benchmarks against real devices and GPGPU-Sim. The results show that PPT-GPU is within 10% compared to real device(s). In summary, PPT-GPU predictions are accurate and highly scalable, where PPT-GPU is 160x faster, on average, than GPGPU-Sim.

We are working to model GPU caches similar to [20], [21]. In addition, we are working to model divergence, power consumption, and will use more recent GPUs as well as integrate PPT-GPU into regular PPT to model CPU-GPU interactions.

ACKNOWLEDGEMENT

The authors would like to thank the anonymous reviewers for their feedback which improved the quality of the paper. We would also like to thank the members of the PEARL laboratory at NMSU. Parts of this research used resources provided at the Los Alamos National Laboratory Institutional Computing Program, which is supported through the U.S. Department of Energy (DOE) National Nuclear Security Administration (NNSA) under Contract No. DE-AC52-06NA25396. Computations were run on Darwin, a research computing heterogeneous cluster. Any opinions, findings, and/or conclusions expressed in this paper do not necessarily represent the views of the DOE or the U.S. Government.

REFERENCE

[1] S. Hong and H. Kim, "An analytical model for a gpu architecture with memory-level and thread-level parallelism awareness," in *ISCA*, 2009.

[2] J. Sim *et al.*, "A performance analysis framework for identifying potential benefits in gpgpu applications," in *PPoPP*, 2012.

[3] Y. Zhang and J. D. Owens, "A quantitative performance analysis model for gpu architectures," in *HPCA*, 2011.

[4] A. Bakhoda *et al.*, "Analyzing cuda workloads using a detailed gpu simulator," in *ISPASS*, 2009.

[5] G. Chennupati *et al.*, *Performance Prediction Toolkit (PPT)*, Los Alamos National Laboratory (LANL), 2017, https://github.com/lanl/PPT.

[6] N. Santhi *et al.*, "The simian concept: Parallel discrete event simulation with interpreted languages and just-in-time compilation," in *2015 Winter Sim Conf.*, 2015.

[7] G. Chapuis *et al.*, "Gpu performance prediction through parallel discrete event simulation and common sense," in *9th VALUE-TOOLS*, 2016.

[8] S. Che *et al.*, "Rodinia: A benchmark suite for heterogeneous computing," in *IISWC*, 2009.

[9] J. Stratton *et al.*, "Parboil: A revised benchmark suite for scientific and commercial throughput computing," *UIUC IMPACT Tech. Rpt, IMPACT-12-01*, 2012.

[10] *NVIDIA Visual Profiler Userss Guide*, 2014. [Online]. Available: http://docs.nvidia.com/cuda/profiler- users- guide/

[11] *Parallel Thread Execution ISA Userss Guide*, 2018. [Online]. Available: www.docs.nvidia.com/cuda/parallel-thread-execution/index.html

[12] M. Andersch *et al.*, "Analyzing gpgpu pipeline latency," in *10th Int. Summer School on Adv. Computer Arch. and Compilation for HP and Embedded Systems*, July 2014.

[13] J. L. Hennessy and D. A. Patterson, *Computer Architecture, Fifth Edition: A Quantitative Approach*, 5th ed. Morgan Kaufmann, 2011.

[14] P. Hall, *"Adaptation of a GPU simulator for modern architectures"*, 2016, mSc. Thesis. [Online]. Available: https://lib.dr.iastate.edu/etd/15712

[15] K. Punniyamurthy *et al.*, "Gatsim: Abstract timing simulation of gpus," in *DATE*, 2017.

[16] E. Konstantinidis and Y. Cotronis, "A quantitative roofline model for gpu kernel performance estimation using micro-benchmarks and hardware metric profiling," *JPDC*, 2017.

[17] S. Williams, A. Waterman, and D. Patterson, "Roofline: An insightful visual performance model for multicore architectures," *Commun. ACM*, 2009.

[18] S. Lee *et al.*, "Compass: A framework for automated performance modeling and prediction," in *ICS*, 2015.

[19] K. L. Spafford and J. S. Vetter, "Aspen: A domain specific language for performance modeling," in *SC*, 2012.

[20] G. Chennupati, N. Santhi, S. Eidenbenz, and S. Thulasidasan, "An analytical memory hierarchy model for performance prediction," in *2017 Winter Simulation Conference (WSC)*. IEEE, 2017, pp. 908–919.

[21] G. Chennupati *et al.*, "A scalable analytical memory model for CPU performance prediction," in *PMBS@SC*, 2017.