A Taxonomy of GPGPU Performance Scaling

Abhinandan Majumdar
Computer Systems Laboratory
Cornell University
am2352@cornell.edu

Gene Wu
Dept. of Electrical and Computer Engineering
The University of Texas at Austin
Gene.Wu@utexas.edu

Kapil Dev
School of Engineering
Brown University
kapil_dev@brown.edu

Joseph L. Greathouse, Indrani Paul, Wei Huang, Arjun-Karthik Venugopal, Leonardo Piga, Chip Freitag, Sooraj Puthoor
AMD Research, Advanced Micro Devices, Inc.
{Joseph.Greathouse, Indrani.Paul, WeiN.Huang, Arjun.Karthik, Leonardo.Piga, Chip.Freitag, Sooraj.Puthoor}@amd.com

Abstract—**Graphics processing units (GPUs) range from small, embedded designs to large, high-powered discrete cards. While the performance of graphics workloads is generally understood, there has been little study of the performance of GPGPU applications across a variety of hardware configurations. This work presents performance scaling data gathered for 267 GPGPU kernels from 97 programs run on 891 hardware configurations of a modern GPU. We study the performance of these kernels across a 5× change in core frequency, 8.3× change in memory bandwidth, and 11× difference in compute units. We illustrate that many kernels scale in intuitive ways, such as those that scale directly with added computational capabilities or memory bandwidth. We also find a number of kernels that scale in non-obvious ways, such as losing performance when more processing units are added or plateauing as frequency and bandwidth are increased. In addition, we show that a number of current benchmark suites do not scale to modern GPU sizes, implying that either new benchmarks or new inputs are warranted.**

I. Introduction

Graphics processing units (GPUs) are an increasingly important class of general-purpose accelerators. Programming systems such as OpenCL™ allow software developers to treat GPUs as a general computational resource, a mechanism dubbed general-purpose programming on GPUs (GPGPU).

GPGPU has seen widespread adoption because of its potential for performance improvements, but GPUs cover a wide range of of sizes, frequencies, and memory bandwidth points. This leads us to ask how GPGPU programs perform across these many configurations.

We study the performance scaling of 97 applications (comprised of 348 samples from 267 separate OpenCL kernels) across 891 different hardware configurations and observe a number of common scaling patterns. Intuitive examples include kernels whose performance relies on the hardware's peak computation rate, and those that are coupled to available bandwidth. We also observe non-intuitive scaling patterns, such as plateaus caused by interconnect bandwidth and performance losses from cache thrashing. Finally, we demonstrate that many benchmarks do not take full advantage of large GPUs.

II. Experimental Setup

We ran our tests on an AMD FirePro™ W9100 discrete GPU, which normally runs 44 CUs (2816 function units) at 930 MHz and has 16 GB of GDDR5 memory across 16 channels that run at 1.25 GHz. Each CU has 16 KB of L1 data cache, while 1 MB of L2 cache is shared across all the CUs. We use the June 10, 2014, beta version of the AMD FirePro drivers, version 2.9 of the AMD APP SDK, and AMD CodeXL version 1.4 to measure the performance of the GPGPU kernels.

We changed a number of the GPU's hardware configuration parameters for our tests. We vary the core frequency between 200 MHz and 1 GHz (inclusive), at 100 MHz increments and change the frequency of the GDDR5 connections from 150 MHz to 1250 MHz (inclusive) in 110 MHz increments, yielding a range of 38.4-320 GB/s. Finally, we vary the number of CUs from 4 to 44 (inclusive), in steps of 4. In summary, we measure each kernel on 891 hardware configurations across a 5× change in frequency, 8.3× change in memory bandwidth, and 11× change in parallel compute resources.

III. Benchmarks and Goals

The goal of this study is to observe how GPGPU kernels perform on GPUs that are configured in different ways. To understand the scaling trends, we explore 97 OpenCL™ applications and have measurements for 267 of their kernels.

We inspect benchmarks from Rodinia v3.0 [2], Parboil [8], OpenDwarfs [6], SHOC [5], Pannotia [1] and a number of applications from the AMD APP SDK and the Phoronix test suite. We also run the exascale proxy applications MiniFE, LULESH, CoMD, and XSBench, a CSR-Adaptive kernel [7], and implementations of a B+Tree search [3] and the graph500 benchmark [4]. Our performance measurements include neither the time spent on the host CPU nor the overheads of copying data from the host to the GPU's memory.

IV. How GPGPU Kernels Scale

A. Compute-Bound Kernels

The performance of compute-bounds kernels, such as *JuliaGPU* from the Phoronix suite, improves with more compute resources, which can come from increasing the active CU count or the core frequency. Memory bandwidth does not affect these kernels because of their relatively small memory footprints and/or their well-cached data. Roughly 38% of kernel samples are compute-bound when frequency is varied, but less than 10% are compute-bound when the CU count is varied since many benchmarks do not scale to large CU counts.

B. Memory-Bound Kernels

Memory-bound kernels are primarily affected by the available bandwidth between the GPU and its DRAM. An example of memory-bound kernel is *readGlobalMemoryCoalesced* from the SHOC *DeviceMemory* microbenchmark. We observed that over 30% of our kernels are primarily memory bound.

The performnace of a subset of these kernels eventually start falling as CUs are added. An example of such a performance peak appears in *lbm* from the Parboil suite. This problem occurs due to destructive interference in the shared L2 cache; as more CUs are added, the chip's parallel working set eventually overflows the cache and causes heightened bandwidth requirements.

C. Balanced Kernels

The performance of balanced kernels, such as *sgemmNN* from the SHOC *GEMM* benchmark, depends on both computational capability and memory bandwidth. Such kernels have some compute-to-bandwidth ratio that maximizes performance. Less bandwidth causes memory stalls, while fewer computational resources causes instruction throughput bottlenecks. About 16% of the total kernels show balanced behavior when scaling CU count and memory bandwidth.

D. Kernels That Do Not Scale

Surprisingly, many of the kernels that we studied do not scale with added CUs nor with added memory bandwidth. This behavior often happens because of programming errors or algorithmic limitations. Kernel *normalize_weights_kernel*, for instance, does not scale with CU count because its hot loop (a prefix sum) is completely serial. Kernel *setZero* of the *swat* benchmark is also unscalable because it has only 13 instructions. As a result, the hardware consumes most of the time assigning workgroup to its CUs. *astar* from OpenDwarfs, and *myocyte* from Rodinia are also unscalable because they do not launch enough work to utilize modern GPUs. Overall, nearly 15% of the kernels we analyzed are unscalable.

E. Performance Plateaus

Another interesting scaling trend is a performance plateau. Here, the kernel scales with additional hardware resources up to a point, whereafter it flattens out regardless of any additional compute and memory resources. *BlackScholes* shows such behavior because, for large input sizes, it allocates its memory in the host space instead of in the device's memory. This causes its performance to be limited by the host-to-GPU interconnect bandwidth. Alternately, the performance of *Nqueens* plateaus because it assigns one workgroup per CU, which is not enough work to hide latency through fine-grain multithreading.

In a related manner, *Bottom_scan* from the Sort benchmark shows multiple "terraces" because it has a limited amount of available parallelism. Its small number of workgroups each takes a roughly equal amount of time to finish, meaning that, unless enough CUs are added to evenly divide the work, the remainder workgroups will prevent any performance improvements. About 18% of the kernels we studied showed some type of performance plateau behavior.

V. Conclusions and Future Work

In this work, we studied how GPGPU kernels scale as hardware configurations change. In total, we examined 348 iterations from 267 kernels which come from 97 benchmarks. We analyzed 891 hardware configurations, comprising a 5× change in core frequency, 8.3× change in memory bandwidth, and 11× difference in CUs.

While many kernels scale in intuitive ways (e.g. increasing performance as more CUs are added), a surprising number of kernels either do not scale at all or their performance eventually plateaus. Roughly 40% of the kernel iterations we studied do not scale to modern GPU sizes. While there are numerous reasons for this lack of scalability, GPU sizes will continue to increase for the foreseeable future. Future studies should look at whether existing benchmarks (and input sets) are representative of the workloads that will be run on future cores.

Future works could study more advanced applications and other hardware configurations, such as cache sizes, double-precision floating point rate, and changes in memory channels rather than frequency. Finally, power and energy studies are equally important, and would complement our current study.

References

[1] S. Che, B. M. Beckmann, S. K. Reinhardt, and K. Skadron, "Pannotia: Understanding Irregular GPGPU Graph Applications," in *Proc. of the IEEE Int'l Symp. on Workload Characterization (IISWC)*, 2013.

[2] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A Benchmark Suite for Heterogeneous Computing," in *Proc. of the IEEE Int'l Symp. on Workload Characterization (IISWC)*, 2009.

[3] M. Daga and M. Nutter, "Exploiting Coarse-grained Parallelism in B+ Tree Searches on an APU," in *Proc. of the Workshop on Irregular Applications: Architectures & Algorithms (IA3)*, 2012.

[4] M. Daga, M. Nutter, and M. Meswani, "Efficient Breadth-First Search on a Heterogeneous Processor," in *Proc. of the IEEE Int'l Conf. on Big Data (IEEE BigData)*, 2014.

[5] A. Danalis, G. Marin, C. McCurdy, J. S. Meredith, P. C. Roth, K. Spafford, V. Tipparaju, and J. S. Vetter, "The Scalable HeterOgeneous Computing (SHOC) Benchmark Suite," in *Proc. of the Workshop on General-Purpose Computation on Graphics Processing Units (GPGPU)*, 2010.

[6] W. Feng, H. Lin, T. Scogland, and J. Zhang, "OpenCL and the 13 Dwarfs: A Work in Progress," in *Proc. of the Int'l Conf. on Performance Engineering (ICPE)*, 2012.

[7] J. L. Greathouse and M. Daga, "Efficient Sparse Matrix-Vector Multiplication on GPUs using the CSR Storage Format," in *Proc. of the Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, 2014.

[8] J. A. Stratton, C. Rodrigues, I.-J. Sung, N. Obeid, L.-W. Chang, N. Anssari, G. D. Liu, and W. W. Hwu, "Parboil: A Revised Benchmark Suite for Scientific and Commercial Throughput Computing," University of Illinois at Urbana-Champaign, Tech. Rep. IMPACT-12-01, March 2012.