Andersch, M., Lucas, J., Álvarez-Mesa, M. A., & Juurlink, B.

On latency in GPU throughput microarchitectures

Conference paper | Accepted manuscript (Postprint)
This version is available at https://doi.org/10.14279/depositonce-6884

Andersch, M., Lucas, J., Álvarez-Mesa, M. A., & Juurlink, B. (2015). On latency in GPU throughput microarchitectures. In 2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS). IEEE. https://doi.org/10.1109/ispass.2015.7095801

On Latency in GPU Throughput Microarchitectures

Michael Andersch, Jan Lucas, Mauricio Álvarez-Mesa, Ben Juurlink
Technische Universität Berlin
http://www.aes.tu-berlin.de

***Abstract*—Modern GPUs provide massive processing power (arithmetic throughput) as well as memory throughput. Presently, while it appears to be well understood how performance can be improved by increasing throughput, it is less clear what the effects of micro-architectural latencies are on the performance of throughput-oriented GPU architectures. In fact, little is publicly known about the values, behavior, and performance impact of microarchitecture latency components in modern GPUs. This work attempts to fill that gap by analyzing both the idle (static) as well as loaded (dynamic) latency behavior of GPU micro-architectural components. Our results show that GPUs are not as effective in latency hiding as commonly thought and based on that, we argue that latency should also be a GPU design consideration besides throughput.**

I. Introduction

GPUs as we know them today are intrinsically throughput-focused devices designed to hide microarchitectural latency through heavy use of thread-level parallelism. Over the last few generations of commercial GPUs, throughput has increased substantially (GT200 [2008]: 933 GFLOPS, GK110 [2013]: 5045 GFLOPS) as a result of both architectural innovations and advancements in manufacturing technology.

With this work, we take a step back and question the throughput-only focus in the design of GPU architectures. Consequently, we work towards a better understanding of latencies in GPUs by conducting a multi-step GPU microarchitecture latency analysis. In the first step, we determine and interpret the values of global memory latencies in multiple modern NVIDIA GPU microarchitectures. Afterwards, we employ the GPGPU-Sim [1] performance simulator to investigate how global memory latency behaves in GPUs when executing diverse real-world programs and, consequently, discuss how latency and performance are related in throughput architectures such as GPGPUs.

Unit	Tesla GT200 [3]	Fermi GF106	Kepler GK104	Maxwell GM107
L1 D$	×	45	30	×
L2 D$	×	310	175	194
DRAM	440	685	300	350

TABLE I: Latencies of memory loads through the global memory pipeline over four generations of NVIDIA GPUs.

II. Static Latency Analysis

To generate the results, a single active thread chases pointers through the global memory space while varying both the stride as well as footprint of the data being touched. Readings of the clock register yield an overall timespan for the entire traversal. Then, per-access latency is computed for each combination of stride and footprint [3]. Table I summarizes the obtained latency values.

These experiments were performed on three different GPUs, a GF106 chip derived from the NVIDIA Fermi architecture, a GK104 chip derived from the NVIDIA Kepler architecture and a GM107 chip derived from the NVIDIA Maxwell architecture. In addition, the results are extended by corresponding data from a similar analysis conducted by Wong et al. [3] on a GT200 chip from the NVIDIA Tesla architecture. All results are given in clock cycles in the clock domain of the execution hardware, i.e. the hot clock frequency.

As the table shows, the latency properties of the global/local memory space have undergone dramatic changes over the four generations of GPUs analyzed here. Starting with the Tesla architecture, accesses to global/local memory were uncached and thus, the minimum latency one could expect when using global memory instructions was the DRAM latency. With the Fermi architecture, two levels of caching were introduced into the global/local memory pipeline with hit latencies of 45 and 310 clock cycles. Fermi's DRAM access latency is gigantic at almost 700 clock cycles. While the L1 cache hit latency seems relatively low in context of the L2 and DRAM access latencies, it is enormous from a *CPU* designer's point of view. For example, on Intel's Haswell microarchitecture, even the L3 cache hit latency of 36 cycles is lower than Fermi's L1 data cache hit latency despite Haswell's L3 cache being many times larger (up to 8MB L3 vs. up to 48KB L1) [2].

While all latencies have seemingly decreased on Kepler compared to the previous generation, an important parameter of the memory system is not visible in the table: On Kepler, the L1 data cache is accessible only by *local* memory accesses but no longer by *global* memory accesses. As local memory is utilized almost exclusively for thread-private stack data and register spilling, this new behavior of the L1 cache means that most memory accesses that were purposefully introduced by the programmer (i.e. global memory accesses) have a minimum latency of an L2 cache hit (175 cycles).

Finally, on the Maxwell architecture, this trend continues: Whereas the L1 data cache became local-only on Kepler, it has disappeared completely in the Maxwell design. In addition, the access times for both the L2 cache and DRAM have *increased* compared to the Kepler generation, effectively making Maxwell's global/local memory pipeline slower than Kepler's on every level from a latency point of view.

III. Dynamic Latency Analysis

In this section, we follow up the analysis of static latency in the previous section with a discussion of *dynamic* latency, i.e. latency as measured in the GPU during execution of real-world workloads. For this purpose, we employ the GPGPU-Sim GPU timing simulator [1] with additional instrumentation for latency analysis. For simulation accuracy, we choose a pre-validated GPU configuration that is included with GPGPU-Sim and that resembles a GF100 GPU with the NVIDIA Fermi architecture.

A key question raised by the static latency results seems to be *what exactly constitutes the latency of a given memory access*, i.e. where in the memory pipeline (high-latency) memory requests spend most of their time. To find an answer, we modified the simulator to track all memory requests generated by *instructions* (but not those generated by, for example, cache line evictions or instruction cache fills) and to emit timestamps whenever a given memory request moves from one stage of the memory pipeline to the next. With this information, we can construct diagrams showing a breakdown of the lifetime of memory requests into different latency components. The results of this experiment are shown in Figure 1 for a kernel performing breadth-first search.

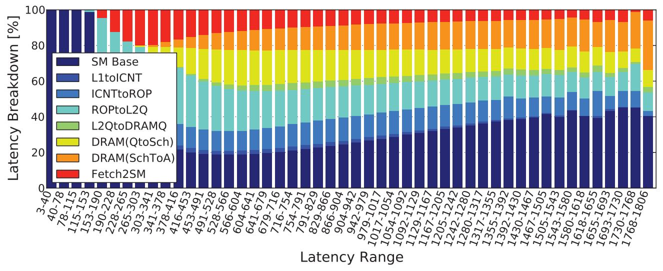

Fig. 1: Breakdown of per-bucket memory fetch latency into pipeline stages for BFS kernel.

In the graph, several latency buckets on the left are entirely filled with SM base time, i.e. the time a request spent in the SM before accessing the L1 data cache. This indicates that all requests in these latency buckets were L1 cache hits. Once we start moving to the right side of the graph, all the memory pipeline stages are present within each bucket, meaning that most of the requests making up the respective buckets missed in at least the L1 and potentially also the L2 caches and went to DRAM for completion.

Overall, the breakdown reveals two important pipeline stages that contribute significantly to the overall latency for loads to global memory, namely the DRAM access scheduling (orange) and L1 miss queue (dark blue) ones. The former means that long-latency requests spend a significant amount of time waiting to be selected for DRAM access, indicating that request latency could potentially be reduced through usage of a different DRAM scheduling algorithm. The latter means that the request spent time traversing a loaded queue - in this case, the one between the SM's L1 cache and the interconnection network. Other workloads similarly showed queueing and arbitration as the two key latency contributors.

Long latencies by themselves are not problematic from a performance standpoint, though. Performance only suffers once latency becomes *exposed*, i.e. cannot be hidden through the execution of other independent work from the same or other in-flight threads. We employ GPGPU-Sim to determine the fraction of load instruction latency that is exposed to investigate how often this occurs in our BFS example workload.

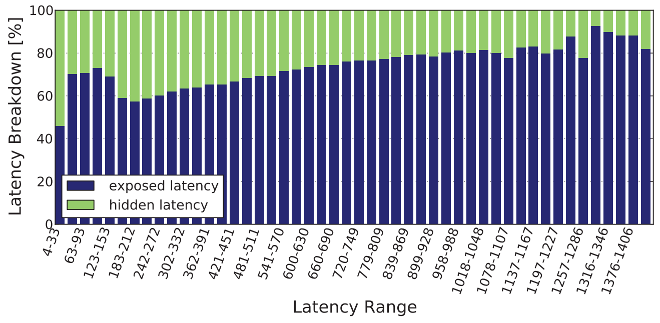

Fig. 2: Breakdown diagram showing wich fraction of global memory load latency was exposed during execution of BFS.

The results of the analysis are shown in Figure 2. Once again, we classify dynamic instructions into buckets depending on their overall latency, and then compute hidden and exposed latency percentages within each bucket. The figures show that latency is, in fact, performance-critical in BFS. The fraction of latency that is exposed is significant, sometimes close to 100% and more than 50% for most of the global memory load instructions.

IV. Conclusions

While many GPU architects seek ways to increase throughput performance and programmability, the role of microarchitecture latency in GPU designs has not received much attention to this point. In this work, we have conducted an evaluation and analysis of latency, both static and dynamic, in a multitude of past and current GPU processor designs.

In the static latency analysis, the last four generations of NVIDIA GPU architectures (Tesla, Fermi, Kepler, and Maxwell) were the subject of microbenchmarking to determine the latencies of the important global memory pipeline. Interestingly enough, the results showed that the latency of said pipeline has *increased* on newer architectures.

In the dynamic latency analysis, we used a GPU performance simulator and an exemplary workload to determine two key contributors to dynamic memory load latency, queueing and arbitration. Lastly, we showed that latency is performance-critical for this particular workload, even though the architecture it is running on is a throughput architecture.

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 *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2009.

[2] P. Hammarlund, A. J. Martinez, A. A. Bajwa, D. L. Hill, E. Hallnor, H. Jiang, M. Dixon, M. Derr, M. Hunsaker, R. Kumar, R. B. Osborne, R. Rajwar, R. Singhal, R. D'Sa, R. Chappell, S. Kaushik, S. Chennupaty, S. Jourdan, S. Gunther, T. Piazza, and T. Burton, "Haswell: The Fourth-Generation Intel Core Processor," *IEEE Micro*, vol. 34, no. 2, Mar 2014.

[3] H. Wong, M.-M. Papadopoulou, M. Sadooghi-Alvandi, and A. Moshovos, "Demystifying GPU Microarchitecture through Microbenchmarking," in *Proceedings of the International Symposium on Performance Analysis of Systems Software (ISPASS)*, 2010.