RETROSPECTIVE: Memory Access Scheduling

Scott Rixner* William J. Dally†‡ Ujval J. Kapasi† John D. Owens§

*Rice University †NVIDIA ‡Stanford University §University of California, Davis

I. Context

This work was a part of the Imagine Streaming Media Processor project at Stanford University led by Bill Dally. The rest of us were PhD students at the time. The key idea behind the Imagine processor was its bandwidth hierarchy, which was designed to support latency-tolerant streaming applications [O16][1]. The hierarchy had small register files feeding each ALU, a banked stream register file, and a bandwidth-focused memory controller. In this paper, we set out to find ways to maximize the achievable memory bandwidth on real applications running on Imagine using commodity DRAM.

We were unconcerned about memory latency, as streaming applications exhibit abundant parallelism and can tolerate long latencies. Imagine performed explicit stream memory transfers between the stream register file and the DRAM. Computations could not be performed on a stream until it was completely transferred to the stream register file. Therefore, there were a large number of outstanding memory requests during any stream memory access. Furthermore, the latency of any individual access was largely irrelevant and only the overall bandwidth of the transfer would impact performance. We relied on other mechanisms to ensure that dependent memory stream references were issued in dependency order. This opened up a large design space of memory controller policies in which we could reorder operations arbitrarily, as long as we maintained memory ordering to any particular address. We carefully studied DRAM datasheets and set out to define a design space to enable us to systematically explore the design of a memory controller for streaming accesses.

After this paper was published, we patented these scheduling policies and mechanisms [2], utilized them in the Stanford Imagine prototype [4], and extended them at Stream Processors Incorporated [5].

II. Insights

In 2000 we had two primary insights. The first was simple, and obvious in retrospect: dynamic random access memory is not random access memory, but instead has access characteristics that are dependent on the multi-dimensional DRAM organization. We can take advantage of this organization when scheduling memory operations. The second was more subtle: a scheduling approach that is aware of DRAM organization can produce a more desirable overall outcome than a default in-order scheduler. Specifically, the default in-order scheduler minimizes the worst-case latency of DRAM requests. This is certainly a desirable outcome! However, our paper showed, and subsequent work has confirmed and extended, that maximizing achieved aggregate bandwidth, even at the cost of worst-case latency, can improve overall program performance.

These fundamental insights from our paper remain true today. The architecture of DRAM is essentially the same, though the bank-row-column organization characteristic of circa-2000 DRAMs now has a fourth layer of hierarchy: a "bank group" (or "page group"), which adds more complexity to a memory access scheduler but presents no theoretical difficulty. DRAMs themselves are also more complex with more complex timing requirements, but the fundamentals behind memory access scheduling remain the same.

III. Lasting Contributions

Looking back in hindsight, there were three major contributions of this paper that have influenced memory controllers ever since. First, we were the first to suggest the idea of scheduling memory operations in order to improve bandwidth. According to Steven Woo, Fellow and Distinguished Inventor at Rambus, memory controllers used in computing have broadly adopted memory access scheduling. This includes CPUs, GPUs, special-purpose accelerators, and other kinds of modern processing units. While the policies are no longer identical to those that were originally presented in the paper, modern memory controllers have built upon the ideas that we first described. So, the core principles of the paper have held up amazingly well and continue to influence the design of all modern memory controllers.

For instance, GPUs have made use of, and extended, these ideas over the past decades. According to Lacky Shah, one of the architecture leaders at NVIDIA:

> The concepts in the paper about reordering the memory system requests are certainly relevant and important to achieve high bandwidth in the GPU memory system. There are many clients of the GPU memory system making requests that happen to have locality in the DRAM banks.... Gathering the requests and organizing them to achieve high memory utilization allows us to saturate memory bandwidth for many workloads and has a big impact on performance. Over time we have extended these concepts to account for different priorities of the requests so we can have multiple such queues for different traffic classes to achieve both high bandwidth and to avoid starvation of various clients that depend on the latency of the requests.

These ideas are also important for domain-specific accelerators implemented via ASICs, FPGAs, and GPUs [3].

[1] [O..] refers to the bibliography of the original paper.

Domain specific accelerators, such as for machine learning, are designed to exploit the parallelism available in a specific application or domain and to balance the performance of the computation resources and the memory. Memory access scheduling is therefore important to ensure efficient use of expensive external memory bandwidth.

Second, not long after the paper was published, it became clear that the insights and ideas of the paper transcend latency-tolerant media processing. By scheduling memory accesses, both bandwidth and average case latency can be improved, leading to overall performance improvements. In particular, the *first ready* scheduler described in the paper (often cited as FR-FCFS), became the "gold standard scheduler" of the single-core era [1]. This scheduler has been considered a general-purpose scheduler and has been the baseline against which most other memory controller policies are compared.

Finally, the basic organization of the scheduler composed of per-bank request tables, as shown in Figure 4 of the original paper, was widely adopted and is still in use today.

The value of memory access scheduling has increased over time. To support such scheduling, modern DRAMs expose a large number of scheduling parameters that can be used by a memory controller. Unfortunately, however, those parameters are almost entirely hidden to programmers or operating systems. Instead, system vendors or OEMs package a set of scheduling parameters into one or a small number of options that can be selected at boot time. Graphics applications and systems typically choose to optimize for maximum achieved bandwidth. But on other platforms, for instance, a vendor may expose a small set of modes to allow the choice of prioritizing performance or power consumption.

IV. Missed Opportunities

One of the principal trends evident since 2000 is the emergence of multicore processors and workloads that share a memory controller. In hindsight, we should have anticipated the simultaneous execution of multiple tasks, possibly with different priorities, competing for access to memory. We should have described how to handle such situations and how to prioritize accesses. Our work assumed the memory system only needed to satisfy a single application's memory traffic. Modern CPU systems, however, may feature dozens of cores and thus potentially dozens of applications competing for memory bandwidth. Furthermore, modern memory controllers support a notion of priority that allows particular address streams or applications to receive preferential treatment. For example, a screen refresh task on a GPU has hard real-time requirements with undesirable visual consequences if pixels are not read and sent to the display in a timely manner. The multicore revolution was in its infancy, but looking back, it seems obvious that it would be important to provide fairness and prioritization to different memory access streams.

We also should have anticipated the possibility of a request being "starved" for a long period of time and included progress guarantees. Optimizing strictly for bandwidth may potentially prioritize a subset of applications that exhibit locality in their accesses, and thus starve others. Consequently, a memory controller must make scheduling decisions that balance performance implications with ensuring all clients make progress.

Because a memory controller must make scheduling decisions in a very short amount of time, it can realistically consider only a small and finite-sized window of possible transactions. (Our 2000 paper analyzed bank buffer sizes of 4–64 entries.) Supporting many simultaneous applications, then, reduces the number of memory requests considered per application. Our work did not consider how this might impact the effectiveness of the various scheduling policies.

V. Final Thoughts

When we wrote this paper, we were focused on memory bandwidth for media processing. Throughout our work, we follow the the mantra "computation is cheap, communication is expensive". This has helped us to stay focused on the right bottlenecks, as we were in this paper. However, while we knew that memory was and would continue to be an expensive bottleneck, we did not realize how much memory access scheduling would grow in prominence as a key mechanism to help mitigate that bottleneck. In hindsight, the ideas presented in this paper have been relevant to a much wider range of systems for a much longer period of time than we ever could have predicted. We are pleased that this paper has held up so well!

Acknowledgments

We greatly appreciate the opportunity to discuss modern memory controllers with Steven Woo (Rambus) and Lacky Shah (NVIDIA).

References

[1] R. Balasubramonian, "Innovations in the memory system," *Synthesis Lectures on Computer Architecture*, vol. 14, no. 2, pp. 1–151, 2019.

[2] W. J. Dally and S. Rixner, "System and method for re-ordering memory references for access to memory," *US Patent #7,707,384*, Apr. 2010.

[3] W. J. Dally, Y. Turakhia, and S. Han, "Domain-specific hardware accelerators," *Commun. ACM*, vol. 63, no. 7, p. 48–57, Jun. 2020.

[4] U. J. Kapasi, W. J. Dally, S. Rixner, J. D. Owens, and B. Khailany, "The Imagine stream processor," in *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers and Processors*, Sep. 2002, pp. 282–288.

[5] B. K. Khailany, T. Williams, J. Lin, E. P. Long, M. Rygh, D. W. Tovey, and W. J. Dally, "A programmable 512 gops stream processor for signal, image, and video processing," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 1, pp. 202–213, 2008.