Designing GPU Architecture for Memory Bandwidth Reservation

Emir C. Marangoz, Kyoung-Don Kang, Seunghee Shin
The State University of New York at Binghamton
emarang1@binghamton.edu, kang@binghamton.edu, sshin@binghamton.edu

***Abstract*—The trend of growing GPU capacity necessitates the support for concurrent executions of multiple applications in a GPU, since a single application cannot fully utilize hardware resources. Interference between the co-running applications, however, increases non-deterministic timing behaviors. In this paper, we propose an extension of the GPU memory architecture to support bandwidth reservation that provides flexible resource sharing, while reserving the required GPU memory bandwidth for each application. Our experimental results show that our approach supports the required memory bandwidth reservation for each application and improves the performance by 20% on average, compared to strict partitioning of hardware resources.**

I. Introduction

Modern GPUs are adopted by a broad range of applications to accelerate the computation with massive parallelism in an energy-efficient manner [1]–[4]. As the amount of hardware resources in GPUs increases, a single application cannot fully use them, causing the enormous resources to be underutilized. To achieve better hardware resource utilization, simultaneous multi-application execution is becoming more relevant [5]–[9].

Although multi-application concurrency is a promising approach to enhance resource utilization, sharing resources such as caches and memory controller generates unexpected dependencies between applications as the memory requests from one application may interfere with requests from other co-running applications [10]–[16]. Inevitably, such unexpected interference makes the time to process memory requests highly variable. Even if the execution time of an application is well analyzed/estimated, running it together with another application easily invalidates the estimation due to different workload characteristics and potential interference. A common approach for reducing the interference between co-running applications is resource partitioning [17] [18] that exclusively assigns a portion of hardware resources to an application and strictly bounds the each application's resource usage to the assignment. However, such strict GPU partitioning scheme [19] achieves performance isolation at the cost of flexibility.

This paper designs an effective GPU hardware resource reservation scheme that provides a predefined reservation of hardware resources to each application when multiple applications run concurrently. Unlike strict partitioning, our approach allows concurrent applications to dynamically share unused hardware resources to reduce latency while supporting resource reservation for co-running applications. Unlike prior bandwidth reservation that regulates the memory traffic in each CPU core or the CPU interconnect [20], we implement our hardware in the GPU's memory controller to manage the actual memory channel usage. Overall, our design extends the GPU memory architecture to support effective bandwidth reservation and sharing with little overhead.

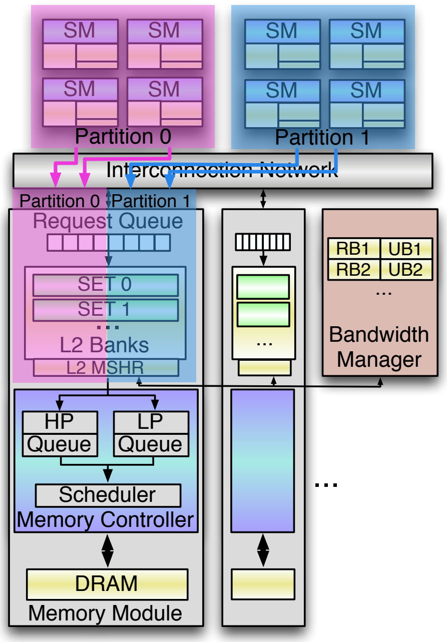

Fig. 1. GPU architecture with bandwidth reservation

II. Bandwidth Reservation in a GPU

We propose extending/adapting the GPU memory architecture for the bandwidth reservations, rather than relying on software implementation while suffering from uncertainties and non-determinism of the black-box GPU hardware. In our design, the GPU memory bandwidth reservation scheme assigns a predefined amount of hardware resources to each application. Then, all the memory requests from an application are marked as high-priority requests when the application still has remaining bandwidth reservation. Once it consumes all the reserved bandwidth in an epoch, any following requests are given low priority. The scheduler in the memory controller always services high-priority requests first. Low-priority requests are serviced when no high-priority requests are waiting for service. In this way, our design supports memory bandwidth reservation and sharing at the same time.

Figure 1 shows our bandwidth reservation framework design that extends the baseline GPU architecture. We insert a hard-

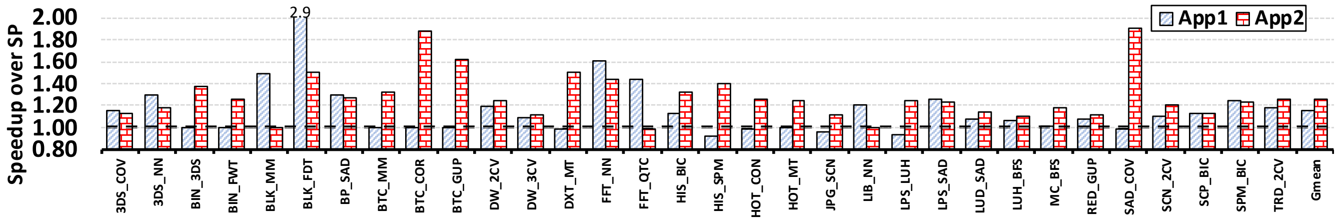

Fig. 2. The speedup of bandwidth reservation over strict partitioning

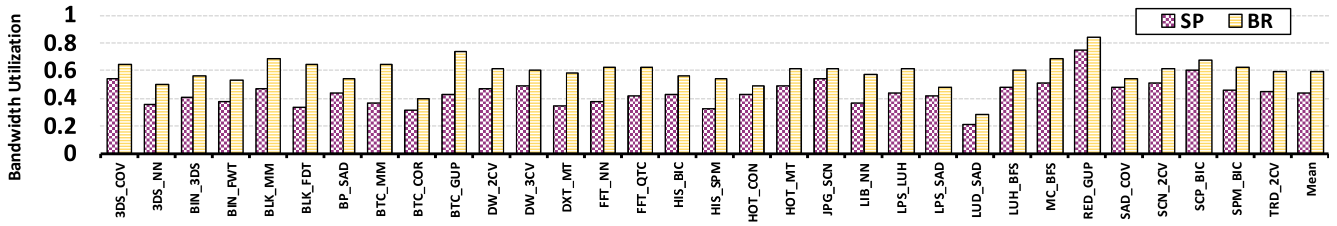

Fig. 3. Bandwidth utilization with bandwidth reservation (BR) and strict partitioning (SP).

ware component, named a bandwidth manager (BM), between the L2 cache and the memory controller in the memory hierarchy to track the bandwidth reservation/consumption status for applications. A memory request reaching the memory controller contacts the BM to inquire about the bandwidth usage status. To do this, the BM has two lists of registers, RB and UB, per application. The RB register holds the value of the predefined bandwidth for an application, and the UB register tracks the bandwidth usage of the application in the current epoch. To implement request prioritization in the memory controller, we divide read/write queues into two queues: a high-priority queue (HP queue) and a low-priority queue (LP queue) for high and low priority requests, respectively. A memory request reaching the memory controller contacts the BM to inquire about the bandwidth usage status.

Memory bandwidth reservation alone cannot support performance isolation because other shared hardware resources, e.g., the L2 shared cache, still impact the performance. Moreover, we observed that such shared resources introduce new performance bottlenecks in our bandwidth reservation framework unless they are properly partitioned. Based on that, we applied strict partitioning on L2 caches, MSHRs, and L2 cache queues.

For our experiments, we have implemented our design in a GPU simulator built upon GPGPU-Sim [21]. We select applications from various benchmark suites including Polybench [22], Rodinia [23], Parboil [24], Ispass2009 [25], SHOC [26], CUDA [27], and LULESH [28]. We evaluate 1) strict partitioning (SP) provided by FGPU [19] and 2) our bandwidth reservation (BR) framework. We run two applications concurrently and measure the performance until one is complete. Thus, the completion time of an application may vary depending on which applications run together.

Figure 2 shows the speedup of our flexible bandwidth reservation against the strict partitioning of FGPU. The two bars in each set represent the speedup of each application, respectively. The final pair of bars shows the geometric mean of all the speedups. We observe that our bandwidth reservation (BR) policy augmented by flexible sharing improves most applications' performance with the overall improvement by approximately 20% on average. In the meantime, the performance of the compute-intensive applications (BIN, BTC, DXT, and HOT) experience little improvement or degradation by bandwidth reservation due to their low dependence on the memory bandwidth. Interestingly, although NN looks like a compute-intensive application in the figure, we found that it is a memory-intensive application with high cache hit rates. NN creates massive memory requests that are mostly serviced in the L1 cache, while the main memory traffic quickly diminishes over time. Due to its high dependence on memory instructions, it still benefits from the improved bandwidth utilization in early execution phases, which can be observed with short applications (3DS and FFT).

We have further investigated a primary reason for the performance improvement achieved by our framework. Figure 3 compares the average bandwidth utilization of BR and SP across all the channels. As expected, the figure shows that BR yields higher bandwidth utilization than SP does in all the workloads. The SP's bandwidth utilization in the figure is only 44% on average, which indicates that SP cannot utilize even half of the given memory bandwidth. However, BR shows 59% utilization that is 15% improvement from the SP's utilization.

III. Conclusions

Multi-application concurrency is a double-edged sword in real-time systems in that it may either enhance performance or decrease it due to increased interference. Meanwhile, the strict partitioning for performance isolation substantially decreases the hardware utilization and performance. Although strict partitioning is effective for some shared resources, the partitioned memory bandwidth leads to significant performance drop. Based on our observations, we extend the GPU memory architecture design to support memory bandwidth reservation and sharing while partitioning other shared resources strictly, all without requiring major changes in software. In an extensive simulation study, our approach supports the specified bandwidth reservations and enhances the performance by 20% on average compared to strict partitioning.

Acknowledgement

This material is based upon work supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korean government(MSIT) (No.2019-0-00533, Research on CPU vulnerability detection and validation). Any opinion, findings, and conclusions or recommendations expressed in this material are those of the authors and should not be interpreted as representing the official views or policies of the Korean government.

References

[1] D. Luebke, M. Harris, N. Govindaraju, A. Lefohn, M. Houston, J. Owens, M. Segal, M. Papakipos, and I. Buck, "GPGPU: General-Purpose Computation on Graphics Hardware," in *ACM/IEEE Conference on Supercomputing (SC)*, 2006.

[2] NVIDIA, "GPU-Accelerated Applications," 2016, http://images.nvidia.com/content/tesla/pdf/Apps-Catalog-March-2016.pdf.

[3] D. Kirk, "NVIDIA CUDA Software and GPU Parallel Computing Architecture," in *International Symposium on Memory Management (ISMM)*, 2007.

[4] J. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. Lefohn, and T. Purcell, "A Survey of General-Purpose Computation on Graphics Hardware," *Computer Graphics Forum*, vol. 26, pp. 80 – 113, 2007.

[5] J. T. Adriaens, K. Compton, N. S. Kim, and M. J. Schulte, "The case for GPGPU spatial multitasking," in *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2012.

[6] M. Lee, S. Song, J. Moon, J. Kim, W. Seo, Y. Cho, and S. Ryu, "Improving GPGPU resource utilization through alternative thread block scheduling," in *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2014.

[7] B. Wu, G. Chen, D. Li, X. Shen, and J. Vetter, "Enabling and Exploiting Flexible Task Assignment on GPU through SM-Centric Program Transformations," in *ACM International Conference on Supercomputing (ICS)*, 2015.

[8] Z. Wang, J. Yang, R. Melhem, B. Childers, Y. Zhang, and M. Guo, "Simultaneous Multikernel GPU: Multi-tasking throughput processors via fine-grained sharing," in *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2016.

[9] R. Ausavarungnirun, V. Miller, J. Landgraf, S. Ghose, J. Gandhi, A. Jog, C. J. Rossbach, and O. Mutlu, "MASK: Redesigning the GPU Memory Hierarchy to Support Multi-Application Concurrency," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2018.

[10] D. Dasari, B. Andersson, V. Nelis, S. M. Petters, A. Easwaran, and J. Lee, "Response Time Analysis of COTS-Based Multicores Considering the Contention on the Shared Memory Bus," in *IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TRUSTCOM)*, 2011.

[11] Lei Liu, Z. Cui, Mingjie Xing, Y. Bao, M. Chen, and Chengyong Wu, "A software memory partition approach for eliminating bank-level interference in multicore systems," in *International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE, 2012.

[12] R. Mancuso, R. Dudko, E. Betti, M. Cesati, M. Caccamo, and R. Pellizzoni, ""Real-time cache management framework for multi-core architectures"," in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2013.

[13] R. Pellizzoni, A. Schranzhofer, Jian-Jia Chen, M. Caccamo, and L. Thiele, "Worst case delay analysis for memory interference in multicore systems," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2010.

[14] N. Suzuki, H. Kim, D. d. Niz, B. Andersson, L. Wrage, M. Klein, and R. Rajkumar, "Coordinated bank and cache coloring for temporal protection of memory accesses," in *IEEE International Conference on Computational Science and Engineering (CSE)*, 2013.

[15] P. K. Valsan, H. Yun, and F. Farshchi, "Taming non-blocking caches to improve isolation in multicore real-time systems," in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2016.

[16] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha, "Memory Access Control in Multiprocessor for Real-Time Systems with Mixed Criticality," in *Euromicro Conference on Real-Time Systems (ECRTS)*, 2012.

[17] S. Kim, D. Chandra, and Y. Solihin, "Fair cache sharing and partitioning in a chip multiprocessor architecture," in *Proceedings. 13th International Conference on Parallel Architecture and Compilation Techniques (PACT)*. IEEE, 2004.

[18] G. E. Suh, L. Rudolph, and S. Devadas, "Dynamic Partitioning of Shared Cache Memory," *J. Supercomput.*, vol. 28, no. 1, p. 7–26, Apr. 2004.

[19] S. Jain, I. Baek, S. Wang, and R. Rajkumar, "Fractional GPUs: Software-Based Compute and Memory Bandwidth Reservation for GPUs," in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2019.

[20] F. Farshchi, Q. Huang, and H. Yun, "Bru: Bandwidth regulation unit for real-time multicore processors," in *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2020, pp. 364–375.

[21] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt, "Analyzing CUDA workloads using a detailed GPU simulator," in *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2009.

[22] L.-N. Pouchet and T. Yuki, "Polybench," 2010, http://web.cse.ohio-state.edu/~pouchet.2/software/polybench/.

[23] 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 *IEEE International Symposium on Workload Characterization (IISWC)*, 2009.

[24] J. A. Stratton, C. Rodrigues, I.-J. Sung, N. Obeid, L.-W. Chang, N. Anssari, G. D. Liu, and W.-m. W. Hwu, "Parboil: A Revised Benchmark Suite for Scientific and Commercial Throughput Computing," *IMPACT Technical Report*, 2012. [Online]. Available: http://impact.crhc.illinois.edu/Shared/Docs/impact-12-01.parboil.pdf

[25] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt, "ISPASS2009 benchmark," 2009, https://github.com/gpgpu-sim/ispass2009-benchmarks/.

[26] A. Danalis, G. Marin, C. McCurdy, J. Meredith, P. Roth, K. Spafford, V. Tipparaju, and J. Vetter, "The Scalable Heterogeneous Computing (SHOC) benchmark suite," 01 2010, pp. 63–74.

[27] Nvidia, "CUDA Toolkit 5.5." [Online]. Available: https://developer.nvidia.com/cuda-toolkit-55-archive.

[28] I. Karlin, J. Keasler, and R. Neely, "LULESH 2.0 Updates and Changes," Tech. Rep. LLNL-TR-641973, August 2013.