Student Research Poster: Slack-Aware Shared Bandwidth Management in GPUs

Saumay Dublish
University of Edinburgh
saumay.dublish@ed.ac.uk

ABSTRACT

Due to lack of sufficient compute threads in memory-intensive applications, GPUs often exhaust all the active warps and therefore, the memory latencies get exposed and appear in the critical path. In such a scenario, the shared on-chip and off-chip memory bandwidth appear more performance critical to cores with few or no active warps, in contrast to cores with sufficient active warps.

In this work, we use the slack of memory responses as a metric to identify the criticality of shared bandwidth to different cores. Consequently, we propose a slack-aware DRAM scheduling policy to prioritize requests from cores with negative slack, ahead of row-buffer hits. We also propose a request throttling mechanism to reduce the shared bandwidth demand of cores that have enough active warps to sustain execution. The above techniques help in reducing the memory latencies that appear in the critical path by increasing the memory latencies that can be hidden by multithreading.

Keywords

GPU; Memory Management; Bandwidth Bottleneck

1. INTRODUCTION

GPUs employ heavy multithreading to overlap the high latency memory operations with concurrent execution. When a *warp* executing on an SIMT core encounters a long latency memory operation, it is de-scheduled and deemed inactive, and a new warp is scheduled from a pool of active warps. In memory-intensive applications, cores often exhaust all the active warps thereby exposing the latency of memory operations. In such a scenario, the shared on-chip and off-chip memory bandwidth appear more performance critical to cores with few or no active warps, in contrast to cores with sufficient active warps. Therefore, when such disparate cores are present during the same epoch of execution, the criticality of the available memory bandwidth varies for different cores. In the existing policy, however, the pending memory request tries to acquire a fraction of the shared memory bandwidth as soon as it is conceived, while being oblivious to the relative criticality of the shared bandwidth across other cores.

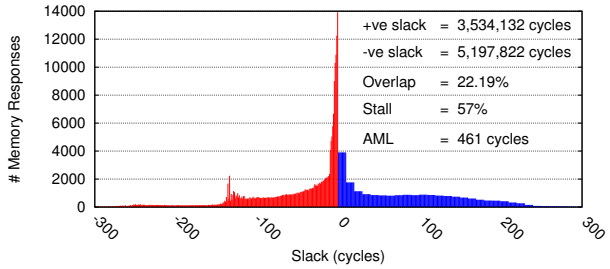

Figure 1: Slack distribution for dwt2D

In addition, memory latencies in memory-intensive applications often exceed the normal memory latencies by a factor of 2-3×. This is because each core employs a *greedy* policy to acquire the shared bandwidth in order to maximize its own bandwidth utilization, thereby excessively depleting the shared bandwidth and causing congestion. It is analogous to the *Tragedy of the Commons*, a problem in economics where a strategy best for an individual in using an unregulated common resource may not yield the most optimal outcome for the group.

In summary, the existing policy of allocating shared on-chip and off-chip memory bandwidth presents two major shortcomings– ❶ allocating bandwidth to cores without regard to their criticality, and ❷ excessive depletion of available bandwidth leading to congestion and therefore high memory latencies.

2. SHARED BANDWIDTH MANAGEMENT

To measure the difference in criticality of shared memory bandwidth on different cores, we record the *slack* in utilizing a cache line that is newly fetched from the lower levels of the memory. We refer to the slack as positive if the memory response arrives sooner than it is required to prevent the core from stalling, and negative if the memory response arrives after the core has stalled. Therefore, positive slack is proportional to the number of active warps on a core whereas negative slack is equal to the cycles for which the core has been stalled since the last issue cycle. In Figure 1, we analyze *dwt2D*, a representative memory-intensive benchmark,

PACT '16 September 11-15, 2016, Haifa, Israel

ACM ISBN 978-1-4503-4121-9/16/09.

DOI: http://dx.doi.org/10.1145/2967938.2971470

and record the slack for each memory response that arrives at the cores. For our experiments, we model a Fermi-like architecture on GPGPU-Sim. We observe that despite high negative slack (also indicated by high stall cycle percentage of 57%), there is a considerable positive slack as well. We also detect the presence of positive and negative slack across different cores, coinciding in the *same epoch of execution*, where the epoch length is 100 cycles. We observe that in such epochs, on average 22.19% of the total negative slack overlaps with positive slack. It indicates that there exists an opportunity to prioritize memory requests from cores that are scarce in active warps (*poor cores*) over cores that have sufficient active warps (*rich cores*). It can lead to a reduction in stall cycles of poor cores without impacting the performance of rich cores. In addition, high average memory latency (AML) of 461 cycles, in contrast to the normal off-chip memory latency of 220 cycles, indicates high level of congestion in the memory system. Therefore, regulating the shared bandwidth allocation also presents an opportunity to reduce the severity of congestion.

To this end, we propose the following two techniques to address the shortcomings discussed above.

❶ *Slack-aware DRAM scheduling.* In epochs where cores with disparity in slack are detected, the proposed scheduler prioritizes requests from cores with negative slack, ahead of row-buffer hits. Therefore, it reduces the memory latencies that lie in the critical path (reducing negative slack) at the cost of increasing the latencies that can still be hidden by multithreading (reducing positive slack). Among cores with varying negative slack, we prioritize cores that are stalled for the longest durations as they have more accumulated pending hits on outstanding memory requests from other warps, allowing more warps to resume execution upon receiving memory response.

❷ *Request throttling.* While the above DRAM scheduling policy regulates the off-chip bandwidth allocation, the congestion in the on-chip memory hierarchy plays an important role in leading to high memory latencies. Therefore, in the proposed scheme, we throttle the rate of sending requests to the shared level for cores that contain high number of active warps, resulting in reduced congestion in the memory system without penalizing performance. In other words, lower the slack of a core, higher the bandwidth allocated to the core.

With the above techniques, preliminary results for *dwt2D* suggest a 7% reduction in negative slack and a 5% reduction in AML.

3. RELATED WORK

Jog *et al.* [2] proposed a DRAM scheduling policy based on core criticality, focusing solely on off-chip bandwidth management. In contrast, we propose a comprehensive policy for regulating on-chip bandwidth (via request throttling) and off-chip bandwidth (via slack-aware scheduling). In addition, they consider a metric that accords equal priority to all stalled cores. On the other hand, we assign different priorities to the stalled cores based on the *slack* metric, to benefit from the accumulated pending hits on outstanding memory requests.

Ebrahimi *et al.* [1] proposed source-throttling in the realm of CMPs with the objective of improving fairness in utilizing shared resources by different applications running on different cores. Similarly, other works have employed fairness oriented schemes to prevent starvation of applications in a co-scheduled environment. For instance, Mutlu *et al.* [3] proposed batch scheduling to prevent starvation of non-memory-intensive application over a memory-intensive application when both applications are scheduled together. In contrast, our work presents a memory management policy for threads *within the same application*, running on different cores in a GPU during different phases of execution.

4. REFERENCES

[1] E. Ebrahimi, C. J. Lee, O. Mutlu, and Y. N. Patt. Fairness via Source Throttling: A Configurable and High-performance Fairness Substrate for Multi-core Memory Systems. In *Proceedings of the Fifteenth Edition of ASPLOS on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XV, pages 335–346, New York, NY, USA, 2010. ACM.

[2] A. Jog, O. Kayiran, A. Pattnaik, M. T. Kandemir, O. Mutlu, R. Iyer, and C. R. Das. Exploiting core criticality for enhanced GPU performance. In *Proceedings of the 2016 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Science, Antibes Juan-Les-Pins, France, June 14-18, 2016*, pages 351–363, 2016.

[3] O. Mutlu and T. Moscibroda. Parallelism-Aware Batch Scheduling: Enhancing Both Performance and Fairness of Shared DRAM Systems. In *Proceedings of the 35th Annual International Symposium on Computer Architecture*, ISCA '08, pages 63–74, Washington, DC, USA, 2008. IEEE Computer Society.