Analysis of Adaptive LLC Caches in GPUs

Summer Project Report

Submitted in fulfillment of the requirements
for the Summer Project

by

Anubhav Bhatla
(Roll No. 200070008)

Under the guidance of

Prof. Virendra Singh

Department of Electrical Engineering
Indian Institute of Technology Bombay

August 2022

Acknowledgement

I express my gratitude to my guide Prof. Virendra Singh for providing me the opportunity to work on this topic.

Anubhav Bhatla
Electrical Engineering
IIT Bombay

Abstract

Graphics Processing Units (GPUs) are widely used to accelerate a wide range of emerging throughput-oriented applications, e.g., machine learning and data analytics. To match the rising computational demands in these fast-growing fields, the number of SMs and the available memory bandwidth have both been steadily increasing with successive generations.

Most GPUs feature a two-level on-chip cache hierarchy in which the first-level caches are private to eac SM while the last-level cache (LLC) is a shared memory-side cache, partitioned in equally-sized slices that are shared by all SMs. This leads to a lower miss rate, but for a sharing-intensive application a private LLC leads to a significant performance advantage because of increased bandwidth to replicated cache lines across different LLC slices.

In this report, I shall anaylyse adaptive memory-side last-level GPU caching proposed by Zhao et al. to boost performance for sharing-intensive workloads by finding a balance between increased LLC bandwidth and increased miss rate under private caching.

Contents

List of Figures

Chapter 1

Introduction

1.1 GPU Architecture

The *graphics processing unit*, or GPU, has become one of the most important types of computing technology, widely used in throughput-oriented applications, e.g. machine learning and data analysis. Due to such high computation demand, GPUs are built with an increasingly large number of *streaming multiprocessors* (SMs).

Each of these SMs accomodate a level-1 instruction cache with its associated cores. Typically, one SM uses a dedicated L1 cache and a shared L2 cache, partitioned into equally-sliced slices and accessed via the *Network-on-Chip* (NoC). While countless research efforts have optimized the cache hierarchy for multicore CPUs, there is a fundamental difference between optimizing the cache hierarchy for a GPU versus a CPU. Unlike latency-sensitive CPUs, GPUs desire high bandwidth access to application data. The traditional CPU solution of designing a banked, large shared LLC can create bandwidth bottlenecks for a GPU. This is especially the case when multiple SMs concurrently access shared data in the shared LLC.

Typically, GPU workloads have large read-only data footprints which leads to a severe performance bottleneck if multiple SMs concurrently access the same shared data. A potential solution to this bandwidth problem is to use a *private LLC*, i.e. replicating shared data across different LLC slices. But this leads to higher cache misses because of cache line replication.

1.2 Shared vs Private LLC

In the shared LLC organization, an LLC slice is shared by all the SMs, see Figure 1.2. The LLC slice for a given cache is determined by a few address bits. Collectively, all LLC slices associated with a given memory controller cache the entire memory address space served by the memory controller.
In the private LLC organization, an LLC slice is private to a cluster of SMs. An LLC slice caches the entire memory partition served by the respective memory controller for only a single cluster of SMs. The LLC slice for a cache line is thus determined by the cluster ID.

Figure 1.2 shows the performance for a private LLC organization normalized to that for a shared LLC organization across three categories of workloads: shared cache friendly, private cache friendly and shared/private cache neutral.

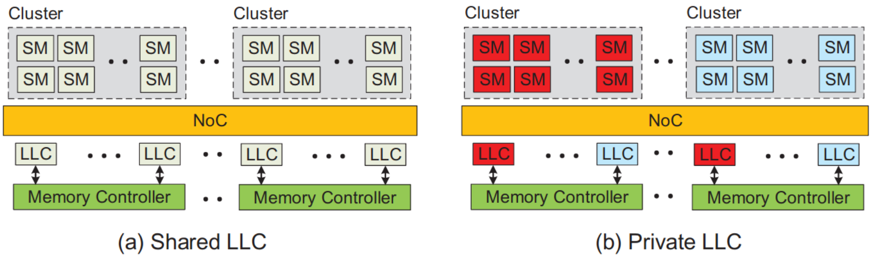
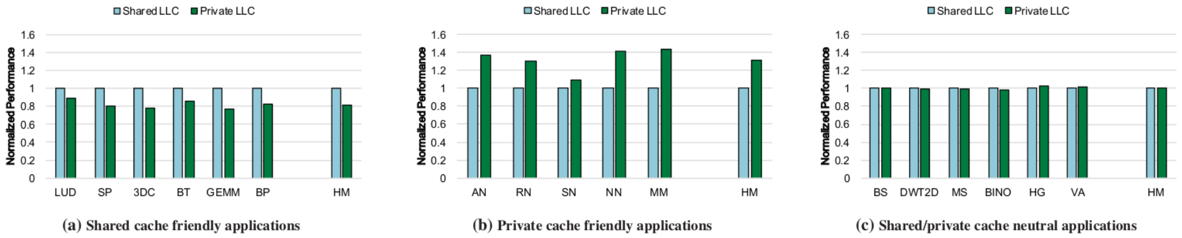


Figure 1.1: Shared and Private memory-side LLC organization in GPUs

We observe that for *private cache friendly* workloads, a bandwidth bottleneck is created due to multiple SM clusters trying to access the same cache lines simultaneously. For such workloads, replicating the cache lines across LLC slices in a private LLC organization leads to better performance under private caching.
The *shared cache friendly* applications, on the other hand, suffer from a 3× increase in miss rate under the private LLC organization, ultimately leading to lower performance. Finally, the *shared/private cache neutral* applications lack inter-cluster locality and hence are neutral to the LLC organization.



(a) Shared cache friendly applications

(b) Private cache friendly applications

(c) Shared/private cache neutral applications

Figure 1.2: Normalized performance for a shared versus private LLC

Chapter 2

Literature Survey

2.1 Decoupled L1 Caches

GPUs use per-core private local L1 caches which leads to low per-core L1 bandwidth utilization while the L2 cache and memory are heavily utilised. Ibrahim et al. [1] proposed a new *DC-L1 (DeCoupled L1) cache* wherein the L1 cache is separated from the GPU core. This helps in reducing replication across the L1s and increases their bandwidth utilisation. We can now aggregate the DC-L1 caches into bigger cache (maintaining the total cache capacity), wherein each DC-L1 cache is accessed by a cluster of GPU cores. The authors further go on to propose a *Clustered DC-L1* design wherein the DC-L1 caches are grouped into clusters and shared cache organization is implemented within this cluster to avoid cache line replication in the cluster and also reduce replication across all the DC-L1s.

Chapter 3

Proposed Idea

The authors propose adaptive last-level caching [2], an LLC organization that dynamically reconfigures the LLC from a shared to private configuration, and vice-versa. They have also adopted a two-stage hierarchical crossbar which offers similar performance to concentrated and full crossbar but is much more area and power efficient.

3.1 NoC-Enabled Adaptive LLC

3.1.1 Coherence Implications

GPUs exploit software-based coherence to circumvent the need for hardware coherence support. To support adaptive last-level caching, the LLC needs to support a *write-through policy when configured as a private cache*, and when the L1 cache is flushed, the private LLC needs to be flushed as well.

Another concern for the private LLC is how to deal with the global memory atomic operation which is handled by the *raster operations (ROP) unit* in the LLC. One solution is to set a small size LLC near the memory controller that is always shared by all SMs to handle atomic operations. Alternatively, one could dynamically opt for the shared LLC organization if the workload contains global atomic operations.

3.1.2 Dynamic Reconfiguration

During a transition between the two LLC organizations, we first need to *stall the SMs* and wait until there are no more in-flight packets in the NoC and memory system. We then need to write back the dirty cache blocks and flush the LLC in the LLC to main memory when transitioning from shared to private caching. Finally, we *power-gate or power-on the MC-routers* to engage private and shared caching, respectively.

The *overhead for these reconfiguration steps is a couple hundred cycles*, which is minimal.

3.1.3 Dynamic Profiling

Profiling estimates LLC miss rate and bandwidth consumption of the private LLC organization while executing under a shared LLC. Each profiling phase takes 50K cycles. We initiate a profiling phase every 1M-cycle epoch or whenever a new kernel comes in. The runtime *overhead for dynamic profiling is limited to 0.8% on average.*

3.1.4 Multi-Program Support

If the co-executing applications prefer a different LLC mode, we can have different LLC modes may be employed for the different applications as they co-execute on a multi-tasking GPU. We can map the different programs across the different clusters as illustrated in Figure 3.1.4 to uniformly distribute the workload across the different clusters while enabling the co-executing applications to access the entire LLC capacity.

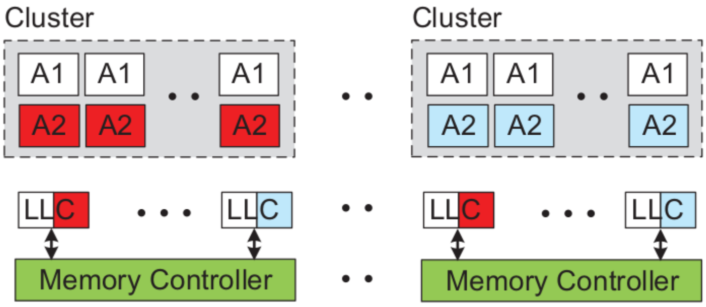


Figure 3.1: Multiprogram Support

3.2 Reconfigurable H-Xbar Support

For a shared LLC configuration, all input ports in the MC router need to be connected to any of the output ports, i.e. the MC router needs to be powered on. On the other hand, in case of a private LLC configuration the input port needs to be connected to its corresponding output port, i.e. the MC router needs to be bypassed. This reconfiguration between power-gating and powering-on incurs an *overhead of a couple tens of cycles.* Figure 3.2 illustrates the re-configurable MC-router to support adaptive last-level caching in H-Xbar.

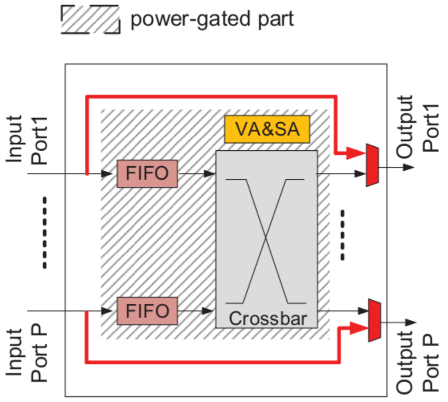


Figure 3.2: MC-router architecture in the reconfigurable H-Xbar

3.3 Reconfiguration Rules

The profiling information is used to decide whether to switch to a private LLC organization or continue with a shared organization. For this we use the following reconfiguration rules:

- **Rule #1** Transition from shared to private if both organizations lead to similar miss rates. This indicates that the application is shared/private organization insensitive. So, we can power-gate the MC routers to save significant power. We switch to private LLC if the miss rate is within 2% of the miss rate in a shared organization.
- **Rule #2** Transition from shared to private if increase in bandwidth overshadows increase in miss rate.
- **Rule #3** Transition from private to shared when a new kernel starts or at the start of a new 1M-cycle epoch.

3.4 Model for Miss Rate and Bandwidth

In our attempt to quantify the trade-off between miss rate and bandwidth, we define a new metric called the *LLC Slice Parallelism (LSP)* which is defined as the sum of all LLC slice accesses (LLC_i) divided by the maximum LLC slice access count for all N slices.

$$LSP = \sum_{i=1}^{N} LLC_i / MaxLLC_i \tag{3.1}$$

LSP can now be used to calculate the overall bandwidth:

$$BW = LLC_{hit} \cdot LSP \cdot LLC_{BW} + LLC_{miss} \cdot MEM_{BW} \tag{3.2}$$

The first term represents the effective LLC bandwidth and the second term computes the effective memory bandwidth.

Chapter 4

Conclusion

4.1 Work done so far

- Studied and reviewed the SIMT Core, Memory systems and the programming model related to GPU architecture.
- Became comfortable with using GPGPU-Sim and simulated various tests and operations on the simulator and analyzed the benchmark outputs received.
- Reviewed literature and analyzing and leveraging decoupled L1 caches.
- Reviewed the use of an Adaptive memory-side LLC organization for improved performance, reduced power and area consumption.

4.2 Future plans

- Review the source code for the GPGPU-Sim simulator, trying to understand the implementation for L2 caches.
- Add necessary components in the code to implement adaptive LLC organization.
- Compare the performance results for a variety of workloads.

4.3 End goal

The end goal for this summer project is to implement Adaptive Memory-side LLC Caching on GPGPU-Sim and then compare the performance results with Shared LLC organization.

References

[1] M. A. Ibrahim, O. Kayiran, Y. Eckert, G. H. Loh, and A. Jog, "Analyzing and leveraging decoupled l1 caches in gpus," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 467–478, 2021.

[2] X. Zhao, A. Adileh, Z. Yu, Z. Wang, A. Jaleel, and L. Eeckhout, "Adaptive memory-side last-level gpu caching," in *Proceedings of the 46th International Symposium on Computer Architecture*, ISCA '19, (New York, NY, USA), p. 411–423, Association for Computing Machinery, 2019.