Adaptive Memory-Side Last-Level GPU Caching

Summer Project Monthly Report - August

Submitted in partial 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

GPUs typically partition the memory-side last-level cache (LLC) in equally-sized slices that are shared by all SMs. Although this leads to lower miss rate, 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 paper [1], the authors propose adaptive memory-side last-level GPU caching to boost performance for sharing-intensive workloads. Adaptive caching balances increased LLC bandwidth against increased miss rate under private caching.

The authors also propose a two-stage hierarchical two-stage crossbar NoC which helps save energy by power-gating and bypassing the second stage if the LLC is configured as a private cache.

Contents

List of Figures

Chapter 1

Introduction

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).
GPUs typically feature two levels of cache hierarchy in which the Level-1 caches are private to the SMs while the Level-2 cache is shared among all the SMs, partitioned into equally-sliced slices and accessed via the *Network-on-Chip* (NoC).
Typically, GPU workloads have large read-only data footprints. This 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.

1.1 Shared vs Private LLC

In the shared LLC organization, an LLC slice is shared by all the SMs, whereas each slice is private to a cluster of SMs in case of a private LLC. Figure 1.1 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.

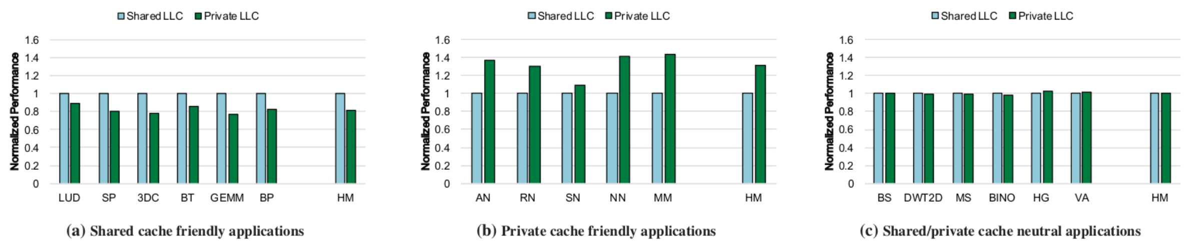

Figure 1.1: Normalized performance for a shared versus private LLC

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.

1.1.1 Adaptive LLC

The observations made above suggest an adaptive memory-side LLC in which the organization changes based on the workload. This reconfiguration from a shared to a private cache can be done easily by dynamically changing the bits used to index the LLC. The target LLC for a shared LLC is determined using the address bits of the request whereas we use the cluster ID for a private LLC.
Although memory-side adaptive caching can be implemented irrespective of the underlying NoC, we can potenitally save energy consumption in the NoC if it is properly co-designed with the rest of the GPU. In the next section, we will look at different GPU crossbar designs, trying to find the best fit for the proposed idea.

1.2 GPU Crossbar Design

1.2.1 Full Crossbar

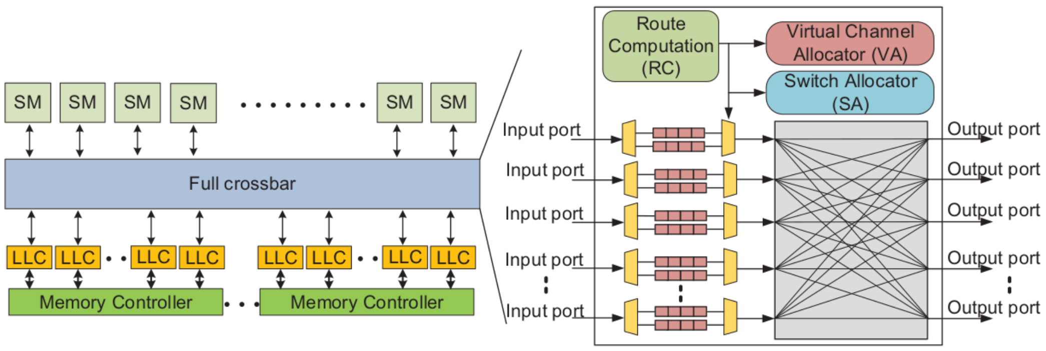

Figure 1.2: Full XBar

Figure 1.2.1 shows the full crossbar, connecting all SMs to the LLC slices. The crossbar only provides connections between SMs and LLC slices and not between SMs or among the LLC slices.

1.2.2 Concentrated Crossbar

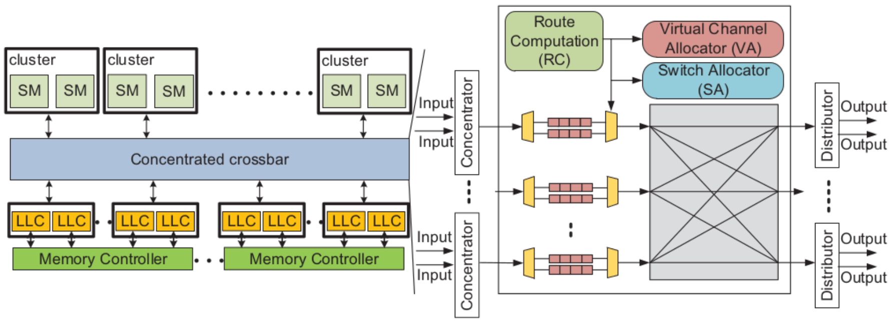

Figure 1.3: Concentrated XBar

To improve cost-efficiency, concentration can be devised where SMs and LLC slices are grouped in a cluster to share one network port through a concentrator and a distributor, respectively. Figure 1.2.2 shows a concentrated crossbar with a concentration of two, which means that two SMs and two LLC slices share one network port.

1.2.3 Hierarchical Crossbar

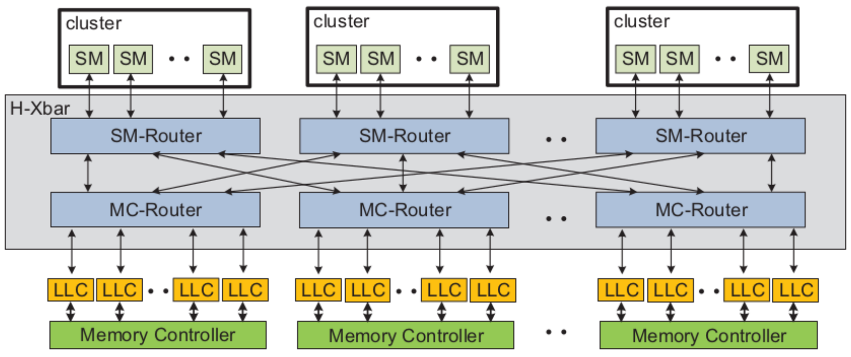

Figure 1.4: Hierarchical XBar

Figure 1.2.3 illustrates the H-Xbar NoC with two stages of routers, the SM-router and the MC-router. The SM-routers connect the SMs to the MC-routers. The MC-routers connect the SM-routers to the LLC slices.
The *Hierarchical Crossbar (H-XBar) achieves similar performance* as a full or concentrated crossbar with the same bisection bandwidth. This is because GPUs are more sensitive to bandwidth than latency, hence the extra hop count for H-Xbar compared to a concentrated or full crossbar does not significantly affect performance.
H-XBar is more area efficient than a concentrated or full crossbar. H-Xbar employs several low-radix crossbars that consume much less chip area than the high-radix crossbar used in the full and concentrated crossbars. This leads to a substantial net NoC area reduction ranging between 62% and 79%.
H-XBar is more power efficient than a concentrated or full crossbar. The short links that connect the SMs and the LLC slices to the SM-routers and MC-routers, respectively, and the long low-bandwidth links between the SM-routers and MC-routers con- sume much less power than the long high-bandwidth links employed in the full and concentrated crossbars. Overall, the hierarchical cross-bar consumes up to 80% less power than a concentrated design.
The Hierarchical Crossbar delivers similar performance while drastically reducing chip area and power consumption as compared to a concentrated or full crossbar. Therefore, the authors decided to use the *H-XBar as the baseline NoC* for their paper.

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. [2] 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. Figure 2.1 shows the design of *Sh40+C10 with 40 DC-L1s and 10 clusters.* Each cluster consists of 8 cores accessing 4 shared DC-L1s via an 8×4 crossbar in NoC#1. The 40 DC-L1s are connected to the 32 L2 slices via four 10×8 crossbars in NoC#2.

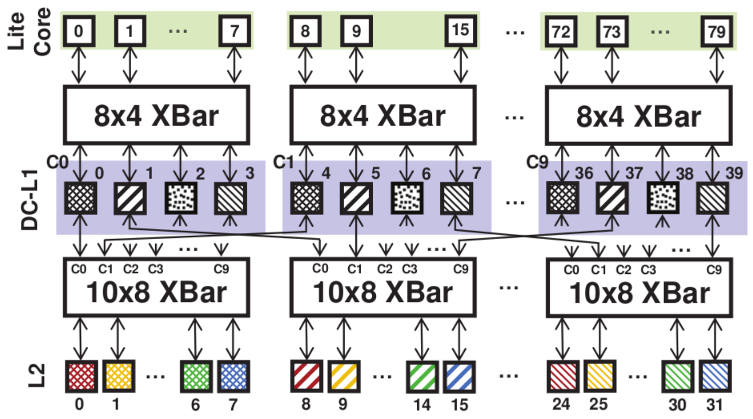

Figure 2.1: Sh40 + C10 Design

The proposed SH40+C10 design provides a *performance improvement of 41%.* For the replication-insensitive applications a 7% drop in performance is incurred. The reduction in the DC-L1 miss rate under Sh40+C10 is higher compared to Pr40 and lower compared to Sh40. Sh40+C10 reduces the NoC static power by 16% compared to baseline but the dynamic power of Sh40+C10 is on average 20% higher because of the high traffic volume in NoC#1. This leads to a *net decrease of 2% in power consumption.* Sh40+C10 improves performance-per-watt and energy efficiency (performance-per-energy), on average, by 29.5% and 95%, respectively. We also *save area* as we use fewer cache ports.

Chapter 3

Proposed Idea

The authors propose adaptive last-level caching [1], 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.

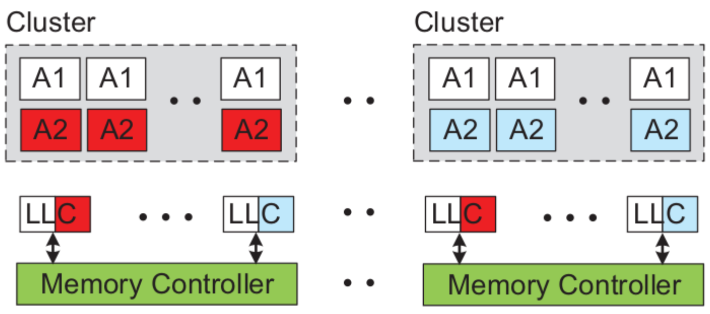

Figure 3.1: Multiprogram Support

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.

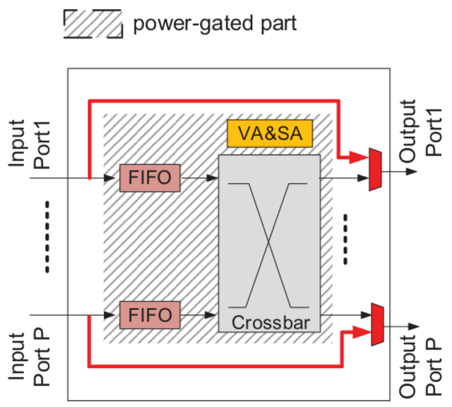

Figure 3.2: MC-router architecture in the reconfigurable H-Xbar

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.

3.5 Evaluation and Conclusion

Experimental results showed an *average performance improvement of 28.1%*, and a maximum improvement of 38.1% for sharing-intensive workloads. Adaptive LLC organization helps in *reducing NoC energy by by 26.6%*, and up to 29.7%, and *system energy by 6.1%* on average, and up to 27.2%, when configured as a private LLC.
Therefore we can conclude that Adaptive caching is an effective and scalable solution to exploit a workload's (lack of) sharing behavior in future GPUs.

Chapter 4

Conclusion

4.1 Work done so far

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

4.2 Future plans

Since I have already reviewed the literature on Adaptive LLC caching, I am now planning to review the source code for the GPGPU-Sim simulator, trying to understand the implementation for L2 caches. Once I am done understanding the necessary part of the source code, I will add necessary components in the code to implement adaptive LLC organization in GPGPU-Sim and finally 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] 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.

[2] 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.