GPGPU L1D- Cache Analysis

CSE 530 Research Project Final Semester Report

Rishabh Jain
Computer Science and Engineering

Pranitha Malae
Computer Science and Engineering

Abstract

In this project we have performed experiments to understand parameters affecting the L1D cache performance and in turn overall performance of GPGPU architectures. We have varied several parameters like number of clusters and size of the cache. and noted various performance metrics like L1D cache miss rate, IPC, replication ratio, and NoC traversal latency. Later we have analyzed obtained values and made conclusions about the effect of these parameters on GPGPU performance. Several modifications have provided clear variations in the performance, but some parameters did not give any conclusive results. We have analyzed the effective usage of cache space by using a parameter called replication ratio. We will propose few methods for effective cache usage. The results of this project have given us an idea about how to proceed further which we will present in the future work section. Our work can be found in this GitHub repository, https://github.com/rishucoding/gpgpu-cache-analysis

1 Introduction

Many applications like Image processing, data processing, audio processing, machine leaning training and testing etc. are using General purpose GPUs (GPGPUs) for faster execution. Performance of GPGPUs play an important role for these applications. Researchers have found that memory wall is an important factor for non-optimal performance in case of GPUs. Multi-level cache usage is the traditional approach to overcome this memory wall problem. Typically, GPUs have a two-level cache architecture. Each core of the GPU has one private L1 cache and all cores share L2 cache. Generally banked L2 cache is used. L2 cache is connected to L1 cache through an interconnection network. Performance of first level cache can have a larger impact on overall memory performance. Improvement in the effective utilization of L1 cache can have positive impact on overall GPGPU performance. Previous research has proved that this traditional cache hierarchy is not using these on-chip caches effectively. So, there is a need to develop new cache design for better performance of GPUs. New designs can only be developed by understanding the problems associated with existing cache design. That is what we tried to do in this project. First, we tried to identify the effect of various parameters on the cache performance. Once we understand the effect of each parameter on the performance, we can develop an optimal cache design. So, main goal of our project is to analyze the results obtained by varying parameters like number of clusters present, and L1D cache size. For each of these variations we have collected information about IPC, L1D cache miss rate, NoC traversal latency and replication ratio. IPC is nothing but number of instructions executed in unit cycle time. If we have more IPC that means we have faster execution of the program. L1D cache miss rate helps to understand the memory latency in the program. If we have less miss rate, then execution of the program can be faster. This also helps to understand cache utilization. Network on Chip (NoC) traversal latency tells us about the communication delay present between cache levels. With the help of this metric, we can make a choice for interconnect network which performs better for most of the applications. We will introduce the replication ratio in the next section. With the help of this analysis, we have presented possible design improvements for effective utilization of L1D cache.

The rest of the paper is organized as follows. Section 2 presents the required background, section 3 presents the GPGPU-sim code exploration and our methodology to incorporate replication counters in the code, section 4 explains about evaluation results, section 5 explains about future work and finally in section 6 we present conclusion.

2 Background

This section explains about the previous work on GPGPUs cache designs and brief overview of GPGPU-sim simulator which we have used for conducting experiments in this project.

2.1 Previous work on Cache designs

Traditional cache hierarchy is not being utilized effectively in GPUs. [1] [2] have proposed new L1 cache designs and have motivated us to pursue cache study. [2] proposes the idea of shared L1 cache instead of traditional private L1 cache and [1] proposes the idea of decoupled L1D cache design. In the case of shared L1 cache design, each L1 cache can store the data related

to a specific address range and is public to all cores. This design requires inter core communication to fetch data related to other address rages which are not present in local L1 cache. In case of shared cache design, we still have caches co-located with the core, but in case of decoupled cache design L1D caches are separated from the core. These decoupled caches can all be aggregated and used, or they can be clustered and shared among particular set of cores. For the decoupled cache design there will be another interconnect network between cores and L1D caches. These two works introduced the idea of replication ratio which we used to understand cache space usage.

Replication Ratio: This is defined as the ratio of L1 misses that can be found in other L1 caches to total L1 misses. This is possible in case of private cache design. In GPUs all tasks are parallelized, and each core executes these tasks individually. When multiple cores are working on same data and if they have private caches, there is a possibility that one data can be present in multiple caches. This is the wastage of expensive on chip cache space. So, these works have come up with shared and decoupled cache designs to minimize replication ratio.

2.2 Overview of GPGPU-sim

GPGPU-sim is the simulator we have used for all our experiments in this project. This is a well-developed and studied simulator for GPGPU architectures. First version of this was introduced in 2007. Docker image of the simulator is available at [12]. The high-level architecture of GPGPU-sim is shown in Figure 1.

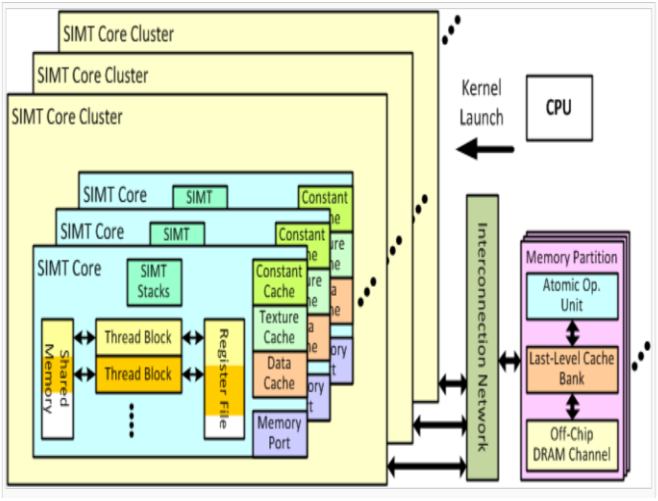


Figure 1: GPGPU-sim architecture

GPGPU-sim contains multiple Single Instruction Multiple Thread (SIMT) core clusters to perform computation. Each of these cluster contains multiple SIMT cores. Each core contains a processor and a scheduler. Each core contains different caches called constant cache, texture cache, data cache. Our focus is on data cache only. Data cache is connected to L2 cache through interconnect network shown as green box in Figure 1. L2 cache is represented as Last- Level Cache bank in Figure 1. Data cache communicates with L2 cache via interconnection network. This L2 cache is connected to off-chip memory which is off-chip DRAM in the GPGPU-sim architecture. When there is a data cache miss in one of the cores of a cluster, it fetches data from L2 cache. So, cache miss time penalty is dependent on interconnect delay and L2 cache access time.

3 Methodology

In this section we will talk about the implementation of GPGPU-sim and the code modifications we made to observe replication ratio.

3.1 Implementation of GPGPU-sim

The GitHub repository of GPGPU-sim is present at [14]. The code hierarchy is shown in figure 2. We have only represented and explained the methods and classes important for our experiments. All these classes contain many other objects and methods in them.

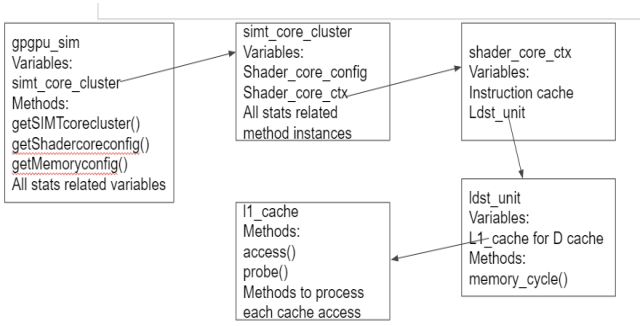


Figure 2: GPGPU-sim code hierarchy

The code is written in C++. The hierarchy presented in figure 2 follows the figure 1 hierarchy.

The top-level class of this design is 'gpgpu_sim' and the object of this class 'gpu' is the global variable which is accessible in all levels of the code hierarchy mentioned above. gpgpu_sim class contains objects of simt_core_cluster class and objects related to all statistics classes. Objects related to these statistics classes are passed down the hierarchy to collect information about each memory access. These statistics contains information about number of cache access of each core L1D cache, total number of instructions executed in this program, ipc, number of cache misses related to L1D cache and L2 cache etc. gpgpu_sim class contains methods which can be used to obtain configuration information of memory and shader cores. Shader core is nothing but the SIMT core mentioned in figure 1. Configuration information is passed to each of these classes by using two separate files called gpgpusim.config and config_ fermi_islip.icnt. gpgpusim.config file contains information about number of cores, number of memory controllers per core, L1D cache policy, L1D cache size, L1D cache design information like number of ways in the cache, size of each cache block etc., L2 cache configuration etc. config_ fermi_islip.icnt file contains information about interconnection network like interconnection network topology, number of virtual channels etc. We could only run butterfly topology interconnect in

version 3.0 of GPGPU-sim, but we could run both butterfly and mesh topologies in version 4.0.1. Butterfly network topology and mesh topologies are shown in figure 3.

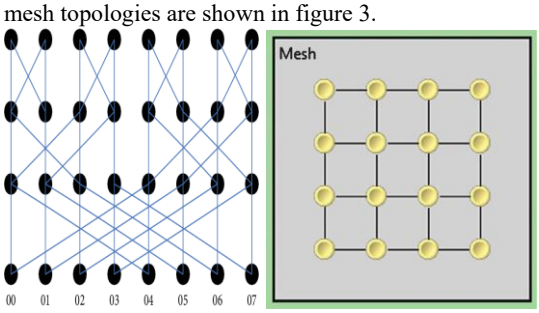


Figure 3: Butterfly and mesh network topology

Next level of hierarchy contains simt_core_cluster class. This class is the representation of SIMT core cluster shown in figure 1. This class contains objects of shader_core_ctx class, shader_core_config class. Shader_core_ctx class is the implementation of SIMT core represented in the figure 1. Shader_core_config is the class which contains the information about shader core like the shader core id, cluster id to which it belongs to etc. shader_core_ctx class is the next in hierarchy and it contains load store unit object of the class ldst_unit. This ldst_unit contains the object of the class l1_cache. L1_cache class is used to create objects of data cache, constant cache and texture cache. All these objects are present in ldst_unit. This l1_cache class contains methods called access and probe. These are the functions related to cache accesses. Probe function is used to know the status of an address like HIT or MISS or RESERVATION HIT etc. Cache misses are handled using miss status holding registers (MSHR) tables. When ever the mshr table is full we will get the RESERVATION MISS status. If we have the address entry in the table already then, we get RESERVATION HIT status. Access function calls the probe function and then modifies the mshr table based on the status received from the probe function and the type of cache access.

Another important function which we need to talk about is the memory_cycle() function. This is the function which performs the cache operations in each clock cycle. This function contains an access queue, which contains the list of pending memory operations to be completed. For each clock cycle we process this access queue and perform the memory operations. This is done using process_memory_access_queue() function. This is the function which contains the object of the cache on which we are performing memory operations and we know the status of cache access in this function. This is the place we have incorporated our code to count the number of data replications present in L1D cache.

3.2 Code modifications

The GPGPU-sim provides many useful statistics related to the performance of L1D cache, interconnect network and overall GPU for the program, but it does not provide information about the replication ratio. So, we have added code to provide the replication ratio present in each benchmark that we are running on GPGPU-sim.

To find the replication ratio, we need to count number of cache misses that are present in other core's caches. For this first we need to know whether a particular cache access is a miss or not and if it is a miss, then we should know the address and core to which this cache belongs to. All this information is present in process_memory_access_queue() function. So, we have added the code for replication ration in this function.

What we have done in the code is that, for every cache miss of an L1D cache object, we run a loop to go over L1D cache objects of all core's caches with the address for which there is a miss. We probe caches of all core's and if we get the status of probe as HIT, then we break the loop and increments the replication count by one. Finally, we find the replication ration by dividing the total replication count with total number of cache misses. This tells how much cache space is wasted because of private caches. In the loop that we are running to go over all core's caches we need to obtain all L1D cache objects. We can get these cache objects by using a global parameter called 'gpu' which is the object of gpgpu_sim class. The code is shown in the following figure.

```
l1_cache* temp =
this->m_core->get_gpu()->getSIMTCluster()[curr_cid]->get_shader_core_object
()[0]->m_ldst_unit->m_L1D
temp_status = temp->m_tag_array->probe(blk_addr,cache_idx);
```

Figure 4: Example code

In this code the temp is the L1D cache object of a core with id curr_id. By varying the value of curr_id we can obtain the L1D cache objects of all cores. One thing to notice here is that we are using the hierarchy mentioned in section 3.1 to get the L1D object. We are using the probe function and not access function to know the status in cache because access function modifies the mshr tables and invokes other methods related to cache writes and reads. We just need to know the status of the cache and we should not access the cache in this case, so we should use probe() function.

3.3 Benchmarks

In this project we have run several benchmarks to generalize the results we obtained. The benchmarks used are from ispass2009 benchmark suit and Tango benchmark suit. From ispass2009 we have used BFS, RAY, STO benchmarks and from Tango we have used Alexnet, Resnet. Tango benchmarks are related to neural network benchmarks which represents the latest application requirements. BFS is the benchmark which represents operations performed on graphs. Many applications use graph processing algorithms, so it is important to incorporate this benchmark. RAY is the application which performs ray tracing in Graphics processing. Other tango benchmarks which do not use GPUs

effectively are LSTM and GRU (Gated neural networks). These are also related to machine learning applications.
We must build binaries of these applications which can be run on GPGPU-sim. To run applications, we have followed the documentation of GPGPU-sim and benchmark suit.

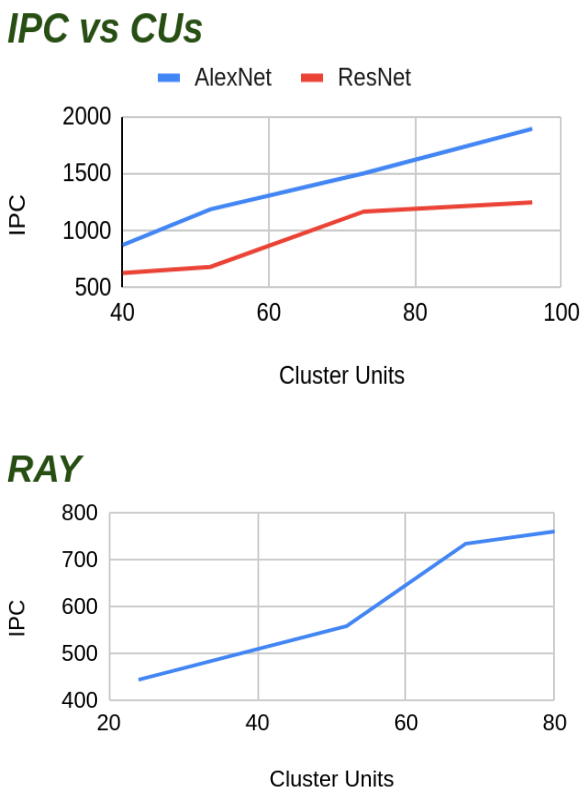
4 Evaluation

4.1 Simulation Strategy

We have used GPGPU-Sim v.3 and v.4 cycle-level simulator[14] for our study. With v.3, we consider GTX480 configuration which is used to run benchmarks: BFS and RAY[14]. With v.4, we consider SM7_TITANV configuration which is used to run benchmarks: AlexNet and ResNet[17]. By default, the L1 data caches are private to each Cluster Unit(CU). Using the simulator we evaluate the impact on IPC, cache_miss_rate and network latency by varying no. of CUs and cache sizes and is discussed in the following subsection.

4.2 Major Results

1. **Role of number of cluster units on IPC:**



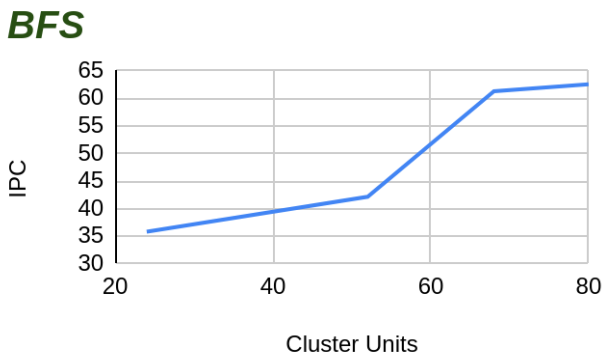


Figure 5: Variation of IPC with number of cluster units

Figure5 describes the change in IPC when sweeping over Cluster Units. There is a clear trend of increase in IPC as CUs grow. This is because more CUs allow more parallelization of application, resulting in faster execution in unit time. Among the 4 benchmarks, AlexNet is able to best utilize multiple cores.

2. **Role of cache size on IPC:**

Figure 6: Variation of IPC with cache size

Figure6 describes the change in IPC when sweeping over cache-sizes. We considered {32,64,128,256}KB for cache-sizes. The IPC seems to benefit with increase in cache size upto a certain value and then saturate. This could be because an increase in cache size helps reduce capacity misses. However, after a certain

point, the working-set of the program fits in the cache and further increase in size doesn't counter other types of misses.

3. **Role of Cache size on cache miss rate:**

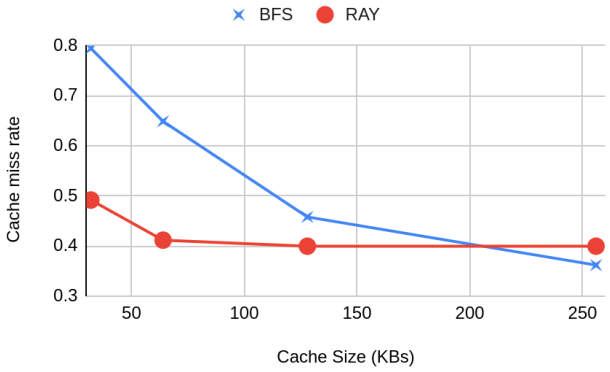


Figure 7: Variation of cache miss rate with cache size

Figure7 describes the change in cache_miss_rate when sweeping over cache-sizes. We considered {32,64,128,256}KB for cache-sizes. To increase cache-size, we changed associativity and number of sets. Block size of 128B is considered. The miss_rate decreases with increase in cache-size. However, the rate of decrease is lower with higher size and could be because compulsory misses can't be handled by changing cache size.

4. **Role of number of cluster units on NoC:**

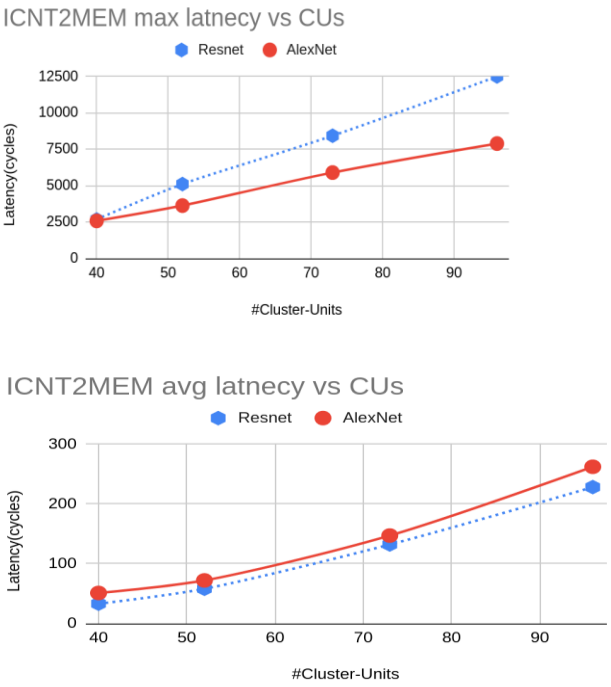


Figure 8: Variation of interconnect latency with number of cluster units

Figure8 describes the change in network traversal latency when sweeping over Cluster Units. A miss in L1 data cache generates a data request for L2 cache. This request travels via request NoC which connects all L1 D$ to L2$ banks. With increase in CUs, injection rate of data-request packets increase leading to high injection traffic. This leads to contention among packets of various CUs and thus an increase in the traversal latency. Both maximum and average latency tends to grow steeply with increase in CUs.

5. **Role of number of cluster units on Replication ratio:**

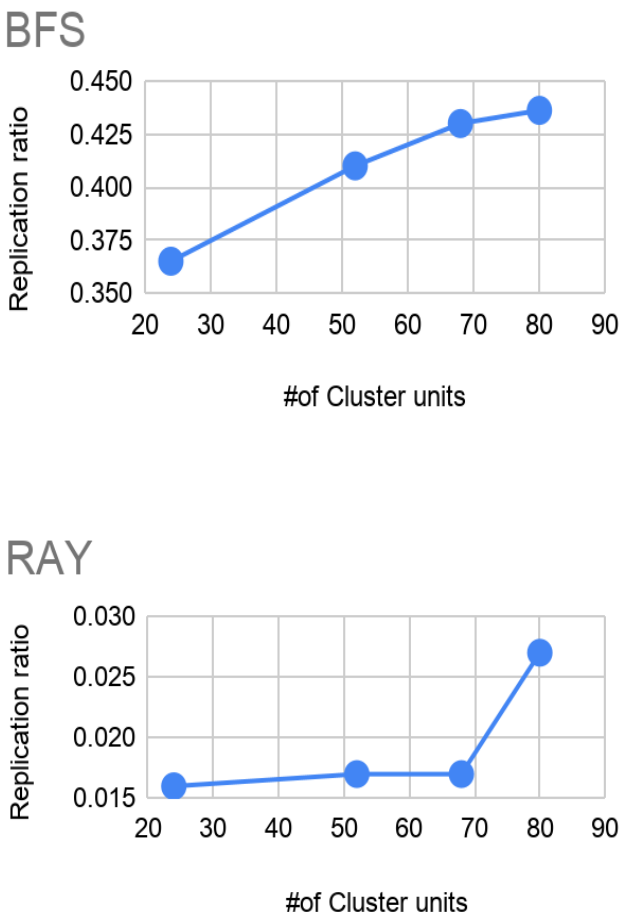


Figure 9: Variation of Replication ratio with number of cluster units

Figure9 describes the change in replication ratio when sweeping over Cluster Units. Replication ratio is (#replications)/(#cache-misses), where replication occurs when a line corresponding to a miss in one L1D$ is found in another core's L1D$. The replication ratio increases with increase in CUs. This could be because an increase in CUs means an increase in private L1D$, thus giving opportunity for cache lines to be found in other places.

6. **Role of Cache size on Replication ratio:**

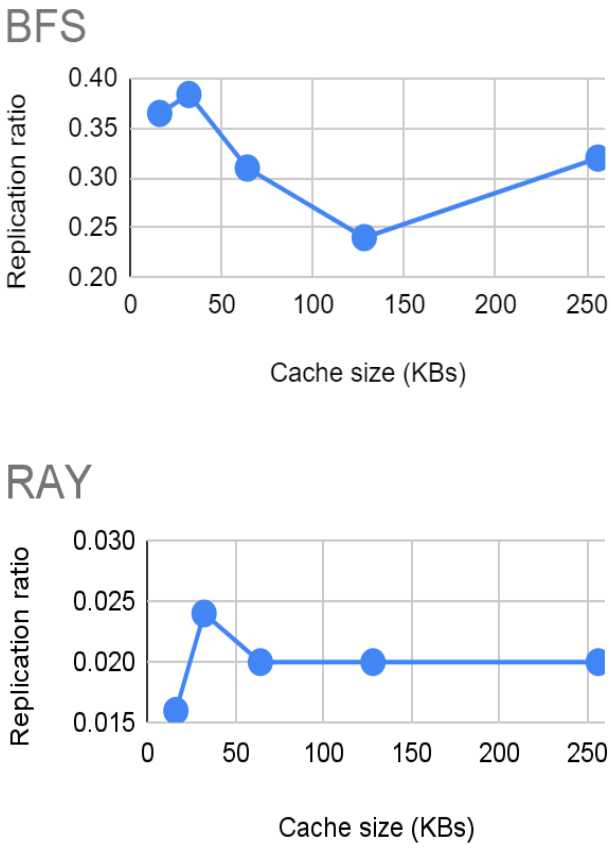


Figure 10: Variation of Replication ratio with cache size

Figure10 describes the change in replication ratio when sweeping over cache-sizes. It is observed that the ratio first increases and then decreases. Replication occurs when CUs share some data which leads to duplication of cache lines in different L1D$s. For applications which have high sharing (eg: use of many global variables) of data, high replication is expected. Also, replication can also occur in applications where CUs depend on each other in a producer-consumer relationship(eg: CU1 generates data which CU2 reads as part of program/algorithm behavior).

7. **Other Observations:**

Among the multiple CUs in a GPU, a nearly uniform distribution of instructions and cache-accesses is observed. This suggests uniform parallelization of programs among the CUs for AlexNet, ResNet, BFS, RAY. However, applications like LSTM and GRU used only 1 and 2 CUs respectectively which suggests heavy underutilization of GPU resources.

5 Future Work

GPGPU-sim simulator is highly configurable and has a modular codebase which models important components like cores, thread scheduling logic, caches, interconnection network, and DRAM. In the future, we plan to do the following:

1. Performing workload characterization with more benchmarks, developing new metrics for evaluation and creating new performance counters similar to replication. This would require us to thoroughly study the simulator codebase and would enable us to obtain new insights. We aim to develop the metrics to find sensitive applications.

2. The work presented in decoupled L1 cache [1] could be further investigated to find better cluster schemes with the aim of mitigation of replication and thus, improving performance. Particularly, a study could be performed to find the right decoupled cache grouping schemes for various applications. If a pattern is established among a category of application, a dynamic clustering mechanism could be developed which allows grouping a set of core's L1D$ which have maximum sharing.

3. An effort in the direction of Hardware-Software co-design could be considered. A study of the program behavior for replication sensitive applications could have patterns where data sharing occurs. These addresses can be marked during compile time and may require new extensions in the ISA. Finally, the GPU can exploit these markers to effectively store data in cache which is bound to be shared.

4. A thorough study of literature work on shared vs private components could help us better understand the existing techniques which address the issue of replication.

5. A study on locality could be performed to understand spatial and temporal behavior of the replicated lines. A threshold value could be used to classify low/high locality associated with a replication block and thus, it could be used to intelligently allow replication.

6. Figure8 establishes a concrete trend of increase in network latency with increase in cluster units. The new generations of GPU architectures like Ampere[16] and Hopper[15] do have increasing core counts and thus, motivates for an efficient interconnection network which is scalable with core count. Adopting the NoC research works which are scalable for many-core designs could be checked for GPUs.

6 Conclusion

In our study, we showed various trends with variations in cluster units, cache sizes over performance metrics like IPC, cache miss rates and network latency. We also study the replication which occurs due to the private nature of L1 data caches. Using the popular GPGPU simulator, our evaluation is performed. As part of our study, we have created the infrastructure: GPGPU-sim and benchmark applications, and plan to continue studying the L1D cache behavior as part of our future work.

7 References

(1) Primary paper:https://adwaitjog.github.io/docs/pdf/decoupledl1-hpca21.pdf

(2) M. A. Ibraim, O. Kayiran, Y. Eckert, G. H. Loh, and A. Jog, "Analyzing and Leveraging Shared L1 Caches in GPUs," in Proceedings of the International Conference on Parallel Architecture and Compilation Techniques (PACT), 2020 - https://adwaitjog.github.io/docs/pdf/sharedl1-pact20.pdf

(3) NVIDIA, "CUDA C/C++ SDK Code Samples," 2011. [Online]. Available: http://developer.nvidia.com/cuda-cc-sdk-code-samples

(4) S. Che, M. Boyer, J. Meng, D. Tarjan, J. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A Benchmark Suite for Heterogeneous Computing," in Proceedings of the International Symposium on Workload Characterization (IISWC), 2009.

(5) A. Danalis, G. Marin, C. McCurdy, J. S. Meredith, P. C. Roth, K. Spafford, V. Tipparaju, and J. S. Vetter, "The Scalable HeterOgeneous Computing (SHOC) Benchmark Suite," in Proceedings of the Workshop on General Purpose Processing Using GPU (GPGPU), 2010.

(6) L.-N. Pouchet, "Polybench: The Polyhedral Benchmark Suite," 2012. [Online]. Available: http://web.cs.ucla.edu/ ~ pouchet/software/ polybench/

(7) A. Karki, C. P. Keshava, S. M. Shivakumar, J. Skow, G. M. Hegde, and H. Jeon, "Detailed Characterization of Deep Neural Networks on GPUs and FPGAs," in Proceedings of the Workshop on General Purpose Processing Using GPU (GPGPU), 2019.

(8) Z. Jia, M. Maggioni, B. Staiger, and D. P. Scarpazza, "Dissecting the NVIDIA Volta GPU Architecture via Microbenchmarking," arXiv, April 2018.

(9) A. Li, S. L. Song, W. Liu, X. Liu, A. Kumar, and H. Corporaal, "Locality-Aware CTA Clustering for Modern GPUs," in Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2017.

(10) S. Dublish, V. Nagarajan, and N. Topham, "Cooperative Caching for GPUs," ACM Transactions on Architecture and Code Optimization (TACO), 2016.

(11) X. Zhao, S. Ma, Z. Wang, N. E. Jerger, and L. Eeckhout, "CD-Xbar: A Converge-Diverge Crossbar Network for High-Performance GPUs," IEEE Transactions on Computers (TC), 2019.

(12) GPGPU-Sim source : http://teaching.danielwong.org/csee217/fall20/gpgpu-sim

(13) Our git repository: https://github.com/rishucoding/gpgpu-cache-analysis

(14) A. Bakhoda, G. Yuan, W. Fung, H. Wong, and T. Aamodt, "Analyzing CUDA Workloads Using a Detailed GPU Simulator," in Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS), 2009

(15) https://en.wikipedia.org/wiki/Hopper_(microarchitecture)

(16) https://www.nvidia.com/en-us/data-center/ampere-architecture/

(17) A. Karki, C. P. Keshava, S. M. Shivakumar, J. Skow, G. M. Hegde, and H. Jeon, "Detailed Characterization of Deep Neural Networks on GPUs and FPGAs," in Proceedings of the Workshop on General Purpose Processing Using GPU (GPGPU), 2019.