Extending L2 Cache in GPUs for Improved CUDA Performance and Area Tradeoff Analysis

Huy G Nguyen
Department of Computer Science
University of Virginia
Charlottesville, Virginia, USA
mpc5ya@virginia.edu

Advisor: Prof. Kevin Skadron
Department of Computer Science
University of Virginia
Charlottesville, Virginia, USA
skadron@virginia.edu

***Abstract*—Modern GPU-accelerated workloads in high-performance computing (HPC), machine learning (ML), and data-intensive simulations often suffer from memory bottlenecks due to limited on-chip cache capacity. In this project, we investigate the performance impact of extending the L2 cache in GPUs for CUDA-based applications using GPGPU-Sim. We explore architectural modifications to increase L2 cache size and assess whether larger cache capacity yields more benefit than improving associativity or prefetching. Using memory-bound benchmarks from the Rodinia suite such as BFS, K-means, Gaussian Elimination, and Needleman-Wunsch, we analyze cache miss behavior, L2 bandwidth utilization, and execution time under various cache configurations. Our results suggest that increasing L2 cache size significantly improves performance in workloads with moderate to high L2 miss rates, particularly when working sets exceed baseline cache capacity. We also discuss the tradeoffs between performance gains and potential overheads in area and energy. This work offers insight into cache design strategies for future GPU architectures and highlights key considerations for tailoring L2 cache extensions to specific CUDA workloads.**

I. Introduction

Modern GPUs are the workhorses of high-performance computing (HPC), powering workloads across scientific computing, machine learning, and real-time graphics. These workloads are characterized by their demand for both massive parallelism and high memory bandwidth. Despite continuous improvements in GPU compute capability, the performance of many data-intensive CUDA applications is increasingly limited not by compute throughput, but by memory bandwidth constraints and inefficient cache utilization.

A key architectural bottleneck lies in the relatively small L2 cache sizes in modern GPUs, which typically range from 4 to 8 MB. For workloads with large working sets and irregular memory access patterns such as those with indirect indexing or graph traversal or the limited L2 capacity often leads to excessive cache thrashing. This results in high L2 miss rates, elevated DRAM traffic, and ultimately, higher memory access latency and energy consumption.

Optimizing L2 cache behavior is especially critical for memory-bound applications. Increasing cache size or improving its design can significantly reduce off-chip memory access and improve overall throughput. In this project, we evaluate the performance impact of extending L2 cache capacity for CUDA applications using GPGPU-Sim. We compare baseline cache configurations with larger L2 sizes, focusing on both performance and tradeoffs in memory efficiency.

Our analysis targets six representative workloads from the Rodinia benchmark suite: Breadth-First Search (BFS), K-means clustering, Discrete Wavelet Transform (DWT2D), Needleman-Wunsch (NW) sequence alignment, Gaussian Elimination, and LU Decomposition (LUD). These benchmarks are chosen for their memory-bound nature and their tendency to exhibit data reuse through spatial and temporal locality. For example, NW accesses and updates a 2D matrix in diagonal wavefronts, while DWT2D operates on image-like data with multi-level hierarchical access. BFS and Gaussian Elimination both rely on structured access patterns over large data arrays, stressing the L2 cache heavily when reused data is evicted prematurely.

In contrast, we also evaluate the Nearest Neighbor benchmark as a compute-bound baseline, where limited memory reuse reduces the relevance of L2 cache extensions. By comparing memory- and compute-bound workloads, our study aims to determine which application characteristics benefit most from increased L2 cache capacity, guiding future architectural optimizations for memory hierarchies in GPUs.

II. Background

In modern multicore processors and GPUs, memory hierarchy design plays a crucial role in bridging the performance gap between high-speed compute units and slower off-chip DRAM. As shown in Figure 1, CPUs typically employ a hierarchical cache system, where each core is equipped with private L1 and often L2 caches, and all cores share a large L3 cache before accessing DRAM. This tiered design reduces contention and allows data reuse across multiple levels of the memory stack.

In contrast, GPUs adopt a flatter and more parallel memory hierarchy to support massive thread-level parallelism. Each Streaming Multiprocessor (SM) is equipped with its own L1 data cache and shared memory, but all SMs share a unified L2 cache. This shared L2 cache acts as the final on-chip buffer before DRAM access and is a key bandwidth-conserving structure for global memory operations. Unlike CPUs, GPUs do not feature an L3 cache, placing even more pressure on

the shared L2 to serve as the primary buffer between compute and DRAM.

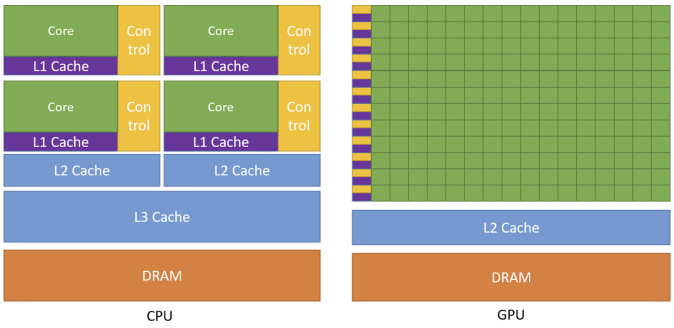

Fig. 1. Comparison of CPU vs GPU memory hierarchies. In GPUs, all SMs share a single L2 cache.

This architectural design introduces both scalability and contention challenges. On high-end GPUs like the NVIDIA Volta architecture, with up to 80 or more SMs, all SMs simultaneously compete for limited L2 cache capacity (typically 4–6 MB). As a result, applications with large working sets or irregular access patterns often suffer from frequent L2 evictions and reduced reuse, leading to high miss rates and greater DRAM traffic.

This becomes particularly problematic for memory-bound applications, where performance is constrained not by compute throughput but by how efficiently data can be delivered to the execution pipelines. Since the L2 cache is the last on-chip opportunity to filter memory traffic, its effectiveness directly affects global memory latency, bandwidth utilization, and energy efficiency.

Understanding how the L2 cache behaves under high contention and diverse memory access patterns is critical for optimizing CUDA workloads. In this project, we explore the impact of extending the L2 cache capacity and analyze how different CUDA benchmarks, with varying memory and compute demands, respond to such architectural changes.

III. Methodology

To evaluate the performance implications of L2 cache extensions on CUDA workloads, we utilize GPGPU-Sim to simulate a Volta-like GPU microarchitecture. This simulator allows fine-grained configuration of the cache hierarchy and provides cycle-accurate measurements of memory behavior. We focus our experiments on six benchmarks: Breadth-First Search (BFS), K-Means Clustering, Gaussian Elimination, LU Decomposition (LUD), 2D Discrete Wavelet Transform (DWT2D), and Needleman-Wunsch (NW) sequence alignment. These applications were selected based on their known L2 cache sensitivity due to high spatial or temporal reuse patterns.

BFS is a graph traversal algorithm in which the frontier and visited node arrays are accessed frequently and benefit from remaining in L2 cache. K-Means Clustering involves repeated access to a small set of centroid data across many data points, making it sensitive to cache reuse. Gaussian Elimination, particularly in its 2D matrix form, operates on overlapping filter windows, causing input elements to be reused across neighboring operations. LU Decomposition repeatedly factors matrix blocks, where intermediate tiles are referenced over multiple phases. DWT2D performs horizontal and vertical filtering passes on images, where retaining scanlines in L2 avoids redundant DRAM accesses. In contrast, Needleman-Wunsch, a dynamic programming algorithm for sequence alignment, processes matrix diagonals, reusing previously computed scores along dependency chains. These access patterns make them suitable candidates to stress the L2 cache.

Our methodology began by running each benchmark across a wide range of input sizes to identify configurations that result in a moderate L2 miss rate, around 70% to 80%. This ensured the cache was neither under- nor over-utilized at baseline, making it easier to observe any performance changes due to cache size adjustments.

We then simulated two strategies for increasing L2 cache capacity. The first involved doubling and quadrupling the number of sets, effectively expanding the total capacity while keeping associativity constant. The second strategy increased the associativity (number of ways per set), which helps mitigate conflict misses without altering the index range. For each strategy, we recorded L2 miss rates, IPC (instructions per cycle), memory stalls, and DRAM traffic using GPGPU-Sim's profiling tools.

To assess hardware feasibility, we complemented our performance analysis with physical modeling using CACTI. For each cache size and associativity configuration, CACTI provided area, latency, and energy consumption estimates. These metrics were used to calculate performance-per-area tradeoffs and determine whether the performance gains justify the increased silicon cost.

This methodology allows us to isolate the effects of L2 cache capacity changes and determine the scenarios where they provide the most benefit. Our analysis highlights which application patterns are most sensitive to L2 size, and whether set-based or associativity-based scaling yields better results in practice.

IV. Design and Implementation

Our experimental framework is based on GPGPU-Sim version 4.0, a cycle-level GPU simulator widely used in academic research. We configured the simulator to model an NVIDIA Volta V100 GPU (SM7 architecture, GV100), which contains 80 Streaming Multiprocessors (SMs), 32 memory partitions, and supports High Bandwidth Memory (HBM2). The simulator was configured using validated configuration files provided by the GPGPU-Sim repository for Volta-based GPUs.

The baseline L2 cache configuration modeled in this study is a unified cache shared by all SMs, organized as follows:

- Each memory partition has 2 memory sub-partitions.
- Each sub-partition includes an L2 cache slice with: 32 sets, 128-byte blocks, and 24-way set associativity.

- This results in a per-sub-partition cache size of: $32 \times 128 \times 24 = 96\,\text{KB}$.
- With 32 memory channels and 2 sub-partitions per channel, the total L2 cache capacity is:

$$96\,\text{KB} \times 32 \times 2 = 6\,\text{MB}.$$

To evaluate the impact of L2 cache size scaling, we implemented four extended cache configurations:

1) **12A (Double Sets)**: Increase the number of sets from 32 to 64, keeping associativity fixed at 24. This results in a per-slice size of 192KB and a total L2 cache size of 12MB.
2) **12B (Double Ways)**: Increase associativity from 24 to 48 ways, keeping 32 sets. This also yields 192KB per slice, resulting in 12MB total.
3) **24A (Quadruple Sets)**: Increase the number of sets from 32 to 128, maintaining 24-way associativity. This provides a total of 24MB of L2 cache.
4) **24B (Quadruple Ways)**: Increase associativity from 24 to 96, with 32 sets. This configuration again results in 24MB total L2.

All modifications were implemented by editing the L2 cache parameters in the GPGPU-Sim configuration file using the $-gpgpu\_cache : dl2$ flag. The simulator's built-in support for variable cache parameters allowed precise tuning of cache structure, including number of sets, associativity, and block size. For consistency, all other architectural parameters (e.g., SM count, DRAM latency, interconnect topology) were held constant across simulations.

This design allows us to isolate the effects of increased L2 capacity and compare whether increasing sets or associativity yields better performance and efficiency across different CUDA workloads.

V. Results and Analysis

A. Finding the Optimal Input Size for Stressing L2 Cache

To ensure that L2 cache extension experiments produce meaningful insights, it is essential to first identify input sizes for each benchmark that sufficiently stress the cache subsystem. An ideal input size yields a moderate L2 miss rate approximately 70% to 80% under the baseline configuration. This range ensures the L2 cache is under realistic pressure but not fully saturated, creating a useful window to observe potential improvements from architectural changes.

For the BFS benchmark, which performs graph traversal over a large adjacency structure, we evaluated several graph sizes ranging from 64K to 16M nodes. As shown in Figure 2, the L2 miss rate increases rapidly with graph size, peaking near 0.9 at 4M and leveling off beyond that. The 2M and 4M input sizes fall within the target miss range, making them ideal candidates for cache sensitivity analysis.

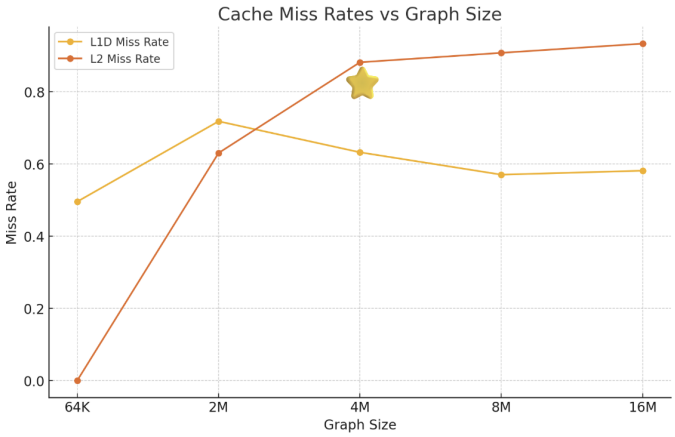

Fig. 2. L1D and L2 Cache Miss Rates vs Graph Size for BFS

This profiling process was repeated for each of the selected benchmarks. For K-Means clustering, an input of 15K data points with a small number of centroids was sufficient to induce a moderate miss rate due to repeated centroid accesses across all points.

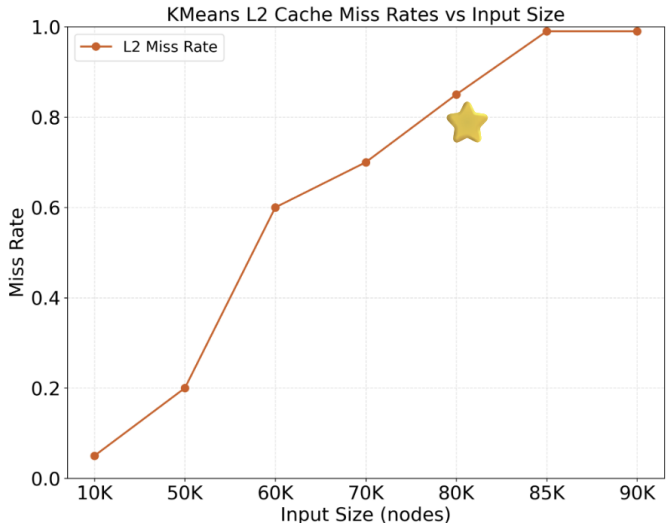

Fig. 3. L1D and L2 Cache Miss Rates vs Input Size for K-Means

For Gaussian filtering and 2D DWT, large 4096×4096 images were required to generate sufficient reuse across overlapping pixel windows and scanlines. The LU Decomposition and Needleman-Wunsch (NW) benchmarks, which operate on full dense matrices and alignment grids respectively, were tested at sizes up to 8192×8192, which produced L2 miss rates within the desired range while remaining tractable for simulation.

Table I summarizes the selected input sizes for each application along with the corresponding L2 miss rates under the baseline configuration. These sizes were subsequently used for all performance comparisons under cache extension scenarios.

TABLE I
SELECTED INPUT SIZES AND CORRESPONDING L2 MISS RATES

Benchmark	Input Size	L2 Miss Rate
BFS	4M Nodes	0.831
K-Means	80K Points	0.852
Gaussian	50K × 50K 2D matrix	0.723
LUD	30K × 30K 2D matrix	0.621
DWT2D	1024 × 1024 pixel image	0.623
Needleman-Wunsch (NW)	4096 DNA length	0.752
Nearest Neighbor (NN)	40K nodes	0.876
Pathfinder	100000 100 20	0.732

B. Impact of L2 Cache Size Extension on Benchmarks

Figure 4 and Figure 5 show the normalized IPC (instructions per cycle) and L2 cache miss rates across all six evaluated benchmarks under five cache configurations: baseline (6MB), 12A (double sets), 12B (double ways), 24A (quadruple sets), and 24B (quadruple ways).

Across the board, all benchmarks demonstrate notable performance improvements with increased L2 cache capacity. However, the degree of benefit and the effectiveness of either increasing associativity or number of sets varies depending on the application characteristics.

BFS and NW exhibit strong gains from associativity-based extensions. Their memory access patterns are less predictable and more scattered (e.g., pointer chasing in graphs and spatial comparisons in sequence alignment), which increases conflict misses under limited associativity. Doubling or quadrupling the number of ways helps reduce these conflicts, resulting in IPC improvements up to 2.17× (NW, 24B) and a 50%+ reduction in miss rates.

KMeans and DWT2D, on the other hand, benefit more from set-based scaling (e.g., 12A and 24A). These applications access a small, repeatedly-used data set such as centroids or image scan lines across a large dataset. Enlarging the number of sets helps accommodate more working set data without interference, leading to the best IPC uplift for KMeans at 2.12× (24A).

Gaussian and LUD show relatively consistent improvements across both strategies. These workloads perform tiled computations on matrices, meaning reuse can benefit from both larger set counts (to hold more tiles) and increased associativity (to mitigate block conflict). Both see over 1.9× improvement under 24A/24B.

Interestingly, **miss rate reduction does not always translate linearly to performance improvement**. For example, while all configurations reduce L2 miss rate substantially (often by over 50%), the IPC gains saturate for some applications beyond a certain cache size. This suggests other bottlenecks (e.g., computation, memory bandwidth, or synchronization overhead) may dominate after a threshold.

These results emphasize that optimal cache configuration is application-specific. Associativity extensions are more effective for workloads with high conflict or irregular memory access (e.g., BFS, NW), while set scaling works better for streaming or spatially-local access (e.g., KMeans, DWT2D). Input size also plays a role: larger datasets naturally stress L2 more, making cache tuning increasingly impactful.

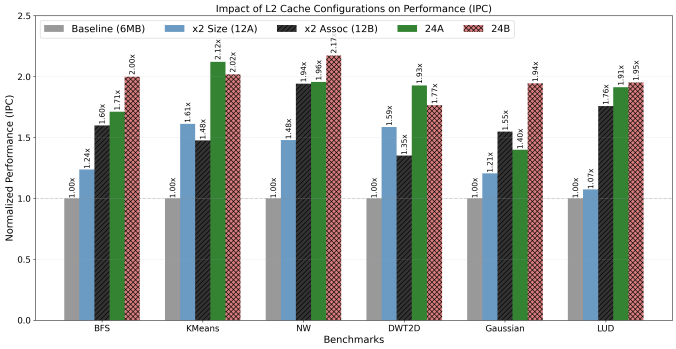

Fig. 4. Impact of L2 Cache Configurations on Performance (Normalized IPC)

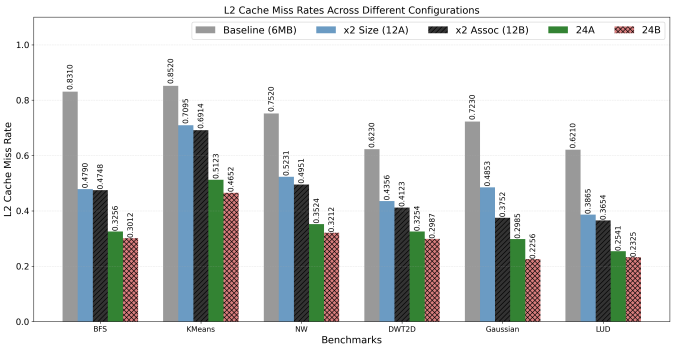

Fig. 5. L2 Cache Miss Rates Across Different Configurations

C. Impact of L2 Cache Size on Compute-Bound Benchmarks

While L2 cache extensions demonstrate significant performance gains for memory-bound applications, their effectiveness is notably diminished in compute-bound workloads. We evaluate this by analyzing two such benchmarks: Nearest Neighbor (NN) and Pathfinder.

Nearest Neighbor (NN) is a classification algorithm that computes the distance between a target point and all other points in the dataset. The memory access pattern is largely sequential and single-use; each data point is loaded once, processed, and not revisited. As a result, there is minimal spatial or temporal data reuse that would allow L2 cache to amortize memory latency, resulting in only marginal IPC gains of around 10–16% even with a 4× cache expansion (Figure 6).

Pathfinder, a dynamic programming benchmark, computes the minimum path cost in a matrix row by row. Although the matrix is reused across iterations, each row is typically processed once before moving to the next, and the working set per row fits within L1 cache. Consequently, increased L2 capacity fails to significantly reduce L2 miss rates or improve throughput.

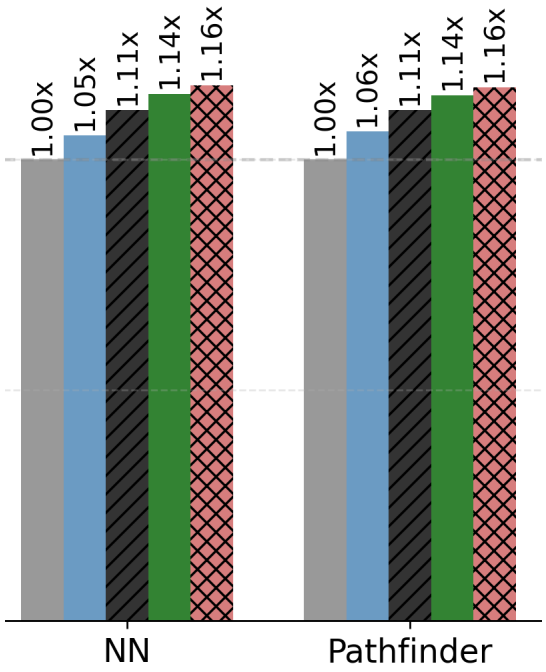

Fig. 6. Normalized IPC for Compute-Bound Benchmarks (NN and Pathfinder)

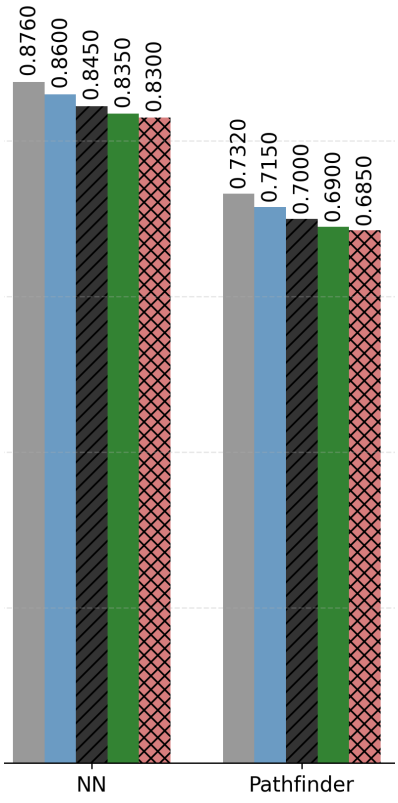

Fig. 7. L2 Miss Rates for Compute-Bound Benchmarks (NN and Pathfinder)

As Figures 6 and 7 show, miss rate reductions in NN and Pathfinder are minor and do not translate into substantial performance improvements. This reinforces that L2 cache size increases are beneficial primarily for memory-bound applications with high data reuse. For compute-bound kernels with little temporal or spatial locality, other optimizations such as instruction pipelining, ILP improvements, or core-level optimizations may be more impactful.

D. Area and Energy Tradeoff

To evaluate the hardware cost of L2 cache scaling, we used CACTI 6.5 to model area and dynamic energy consumption for each cache configuration. CACTI is a widely used tool for modeling cache and memory structures at different technology nodes, providing realistic estimates for timing, power, and physical footprint.

Table II summarizes the projected area and dynamic read energy for five L2 configurations, assuming a 32nm process node and cache block size of 128 bytes. From the baseline 6MB configuration (32 sets × 128B block × 24-way × 32 memory channels × 2 sub-partitions), both doubling strategies either by increasing the number of sets (12A) or ways (12B) lead to a 2.4–2.5× increase in area and nearly 2× higher dynamic read energy. Quadrupling strategies (24A, 24B) approximately double that again, pushing total area to over 2500 mm^2 and dynamic energy beyond 113 nJ.

While 12A and 12B consume comparable energy, their effects on performance vary by benchmark. Applications with high conflict misses (e.g., Gaussian, LUD) benefit more from increased associativity (12B), whereas applications with large working sets and low conflict (e.g., K-Means, DWT2D) perform better with more sets (12A). Similarly, the 24MB configurations (24A/24B) offer diminishing performance returns in some workloads despite substantial increases in resource cost.

These results illustrate a fundamental design tradeoff: L2 cache extension can improve performance by reducing DRAM traffic and memory stalls, but the gains are workload-dependent. If an application only achieves a modest speedup (e.g., 10–15%) with a 2–4× increase in cache area, the investment may not be justified, especially in power or cost-constrained systems. However, for memory-bound applications that exhibit high data reuse, selective L2 scaling can yield up to 40–50% speedup and is a viable optimization.

Ultimately, this analysis supports the need for application-aware L2 cache configuration. The effectiveness of scaling L2 is tightly coupled with the memory access characteristics of the workload, and indiscriminate cache growth may result in poor energy-efficiency.

TABLE II
CACTI-BASED ESTIMATES FOR L2 CACHE CONFIGURATIONS USING SM7_GV100 (32NM, 128B BLOCK)

Configuration	Area (mm^2)	Dynamic Read Energy (nJ)
Baseline (6MB)	512.28	28.50
12A (12MB, double sets)	1278.35	56.74
12B (12MB, double ways)	1231.45	56.97
24A (24MB, quadruple sets)	∼2556.70	∼113.88
24B (24MB, quadruple ways)	∼2579.71	∼114.71

CACTI Configuration for SM7_GV100 L2 Cache Modeling: To model the hardware area and energy characteristics of the Volta V100 (SM7_GV100) GPU's L2 cache, we used CACTI 6.5 with a configuration that mirrors the microarchitectural cache hierarchy. The baseline L2 configuration consists of a total 6MB capacity, implemented as 32 memory partitions, each with two sub-partitions of 96KB (i.e., 96KB × 2 × 32 = 6MB). Each cache block is 128 bytes, with a 24-way set-associative design.

The CACTI input was adjusted accordingly:

- `-size (bytes): 6291456` to match the 6MB capacity
- `-block size (bytes): 128`, aligning with L2 cache line width
- `-associativity: 24`, representing 24-way associativity
- `-UCA bank count: 64`, based on 32 memory channels with 2 sub-partitions each
- `-technology (u): 0.032`, corresponding to a 32nm process node
- `-read-write port: 4`, reflecting concurrent memory access ports
- `-cache type: "cache"`, with `-tag size (b)` set to `"default"`
- `-access mode: "normal"` and optimization target `"ED^2"` for energy-delay squared
- `-Wire signaling: "Global_30"`, `-Wire inside mat: "semi-global"`, and `-Interconnect projection: "conservative"`
- Design objective weights prioritized `cycle time`, with deviation tolerances for other metrics.

These inputs closely reflect the real GPU cache configuration to ensure that the area and energy results from CACTI are representative for tradeoff analysis when exploring double/quadruple cache sizes through either increased sets or associativity.

VI. Conclusions and Future Work

This project explored the performance impact and architectural tradeoffs of increasing L2 cache size in a Volta-class GPU, using GPGPU-Sim for simulation and CACTI for area and energy modeling. We evaluated multiple cache scaling strategies, including increased associativity and increased set count, across six CUDA benchmarks with diverse memory access patterns.

Our findings show that memory-bound applications with significant data reuse, such as BFS, KMeans, DWT2D, and Needleman-Wunsch, benefit from larger L2 caches, especially when their working sets are appropriately sized to stress the cache. However, compute-bound benchmarks like Nearest Neighbor show little to no performance gain, indicating that cache extension is only advantageous under specific access conditions. Additionally, associativity scaling benefited workloads with high conflict misses, while increasing set count was more effective for applications with large spatial working sets.

Future Work

While this study provides a foundational evaluation, several directions remain to deepen and generalize the findings:

- **Detailed Performance-per-Area Modeling:** Extend the CACTI analysis to include full performance-per-area (PPA) and energy-delay-product metrics under real workloads. Incorporating leakage power and wire delay effects would increase modeling accuracy.
- **Exploring Broader Input Ranges:** Evaluate additional input sizes per benchmark to better characterize the cache miss rate thresholds where L2 scaling is effective, and identify working set boundaries.
- **Hardware Configuration Variability:** Simulate GPU variants with different SM/core counts, L1 sizes, and memory bandwidths to determine how L2 cache scaling interacts with other bottlenecks.
- **Software-Level Optimizations:** Investigate the potential of CUDA-level data placement strategies and cache prefetching to complement hardware L2 extension.
- **ML Workloads:** Extend the evaluation to include deep learning training and inference workloads, which often exhibit distinct reuse patterns in convolution and attention layers.
- **Compute-Bound Workload Testing:** Include further analysis on compute-dominated benchmarks to reinforce when and why L2 scaling is ineffective, possibly incorporating Roofline modeling.

These enhancements would provide more comprehensive insight and could bring the work closer to publication quality. Future students can use this framework as a baseline for extending cache hierarchy analysis on modern or emerging GPU architectures, especially in the context of AI and high-performance computing.

VII. Related Work

Prior research has explored various strategies for improving GPU memory hierarchy performance, particularly in addressing cache bottlenecks. This section highlights key contributions relevant to our investigation of L2 cache extensions for CUDA workloads.

Tabbakh et al. [1] proposed a scheduling-aware GPU data management technique that reduces redundant data replication across private L1 caches. By identifying data-sharing patterns among cooperative thread arrays (CTAs), they showed that default round-robin scheduling leads to inefficient L1 cache usage, indirectly increasing L2 traffic due to higher L1 miss rates. Their findings underscore the value of reducing unnecessary cache contention as a motivation shared by our focus on L2 optimization.

Morpheus [2] introduced a novel method of repurposing idle GPU cores as auxiliary last-level cache (LLC) to expand cache capacity dynamically. Although Morpheus targets LLC extensions beyond the L2 level, its results demonstrate that increasing on-chip cache space can yield measurable performance and energy efficiency benefits. This work reinforces the

premise that GPU memory systems can benefit significantly from adaptive cache sizing, which directly informs our study of L2 scalability.

The HMG protocol [3] tackles cache coherence across modern hierarchical multi-GPU systems. While our work focuses on single-GPU architectures, HMG's treatment of coherence scalability emphasizes the rising complexity of memory management in high-parallelism environments. Efficient cache design, even within a single GPU, becomes increasingly critical as memory-bound workloads scale in size and complexity.

In the domain of graphics workloads, Aila and Laine [4] proposed an improved shading cache design to enhance fragment shader reuse and reduce memory bandwidth demands. Though designed for rasterized graphics pipelines, their approach to exploiting spatial locality and reducing redundant fetches is conceptually applicable to CUDA workloads with regular memory access patterns.

Bakhoda et al. [5] presented GPGPU-Sim, a cycle-accurate simulator capable of modeling CUDA kernel execution at microarchitectural detail. This tool serves as the foundation for our experimental evaluation, allowing us to analyze L2 miss rates, memory throughput, and performance trends under varying cache configurations.

Finally, the Rodinia benchmark suite [6] offers a diverse set of heterogeneous workloads that target key parallelism patterns and memory behaviors. We selected six benchmarks, BFS, K-Means, DWT2D, Needleman-Wunsch (NW), Gaussian, and LUD, from Rodinia that are representative of real-world CUDA applications. These benchmarks exhibit varied memory reuse characteristics, making them ideal for evaluating the impact of L2 cache sizing on performance and efficiency.

Together, these prior works highlight the significance of optimizing cache usage in GPU systems. Our contribution differs by focusing specifically on quantifying the performance gains from L2 cache extensions across a range of memory- and compute-bound CUDA workloads using GPGPU-Sim, while considering both architectural and application-level implications.

Acknowledgments

This project was conducted as part of CS 6501: GPU Architecture and Optimization at the University of Virginia, Spring 2025. We would like to thank Professor Kevin Skadron and TA Khyati Kiyawat for their guidance throughout the semester. We also acknowledge the use of UVA's GPU server infrastructure and portal.cs.virginia.edu for running GPGPU-Sim simulations, and we thank the maintainers of Rodinia, GPGPU-Sim, CACTI, and CUDA documentation for providing essential tools and benchmarks for our research.

Disclaimer

This report was prepared within a limited timeframe, and as a result, not all experimental results could be fully included or analyzed. Some evaluations are still ongoing. I acknowledge that portions of the writing may contain grammatical errors or lack clarity, and I appreciate your understanding regarding any shortcomings in the quality of this submission.

References

[1] A. Tabbakh, X. Qian, and M. Annavaram, "Power Efficient Sharing-Aware GPU Data Management," in *Proc. of the IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2021.

[2] S. Sharma, A. Mahalunkar, and R. Das, "Morpheus: Extending the Last Level Cache Capacity in GPU Systems Using Idle GPU Core Resources," *arXiv preprint arXiv:2209.10914*, 2022.

[3] Y. Kim, M. Cho, J. Hwang, and J. Kim, "HMG: Extending Cache Coherence Protocols Across Modern Hierarchical Multi-GPU Systems," in *Proc. of the 55th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2022.

[4] T. Aila and S. Laine, "An Improved Shading Cache for Modern GPUs," in *Graphics Hardware*, 2008, pp. 95–101.

[5] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. W. L. Wong, and T. M. Aamodt, "Analyzing CUDA Workloads Using a Detailed GPU Simulator," in *IEEE Int. Symp. on Performance Analysis of Systems and Software (ISPASS)*, 2009, pp. 163–174.

[6] S. Che, M. Boyer, J. Meng, D. Tarjan, J. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A Benchmark Suite for Heterogeneous Computing," in *Proc. of the 2009 IEEE Int. Symp. on Workload Characterization (IISWC)*, 2009, pp. 44–54.