Application-Specific Autonomic Cache Tuning for General Purpose GPUs

Sam Gianelli, Edward Richter, Diego Jimenez, Hugo Valdez, Tosiron Adegbija, Ali Akoglu
Department of Electrical and Computer Engineering
University of Arizona, Tucson, AZ, USA
{*samjgianelli,edwardrichter,diegojimenez,hugovaldez,tosiron,akoglu*}*@email.arizona.edu*

***Abstract*—Cache tuning has been widely studied in CPUs, and shown to achieve substantial energy savings, with minimal performance degradations. However, cache tuning has yet to be explored in General Purpose Graphics Processing Units (GPGPU), which have emerged as efficient alternatives for general purpose high-performance computing. In this paper, we explore *autonomic cache tuning* for GPGPUs, where the cache configurations (cache size, line size, and associativity) can be dynamically specialized/tuned to the executing applications' resource requirements. We investigate cache tuning for both the level one (L1) and level two (L2) caches to derive insights into which cache level offers maximum optimization benefits. To illustrate the optimization potentials of autonomic cache tuning in GPGPUs, we implement a tuning heuristic that can dynamically determine each application's best L1 data cache configurations during runtime. Our results show that application-specific autonomic L1 data cache tuning can reduce the average energy delay product (EDP) and improve the performance by 16.5% and 18.8%, respectively, as compared to a static cache.**

***Index Terms*—Graphics processing unit, GPGPU, GPU cache management, configurable caches, high performance computing, cache tuning, cache memories, low-power design, low-power embedded systems, adaptable hardware**

I. Introduction and Motivation

A processor's memory hierarchy contributes substantially to the processor's overall energy consumption. In some cases, such as embedded systems' CPUs, the memory hierarchy (specifically, the caches) can contribute more than 40% of the system's total energy [19], [20]. As a result, much optimization research has focused on reducing the cache's energy consumption, without degrading the performance.

The cache is increasingly becoming a source of energy overhead in general purpose graphics processing units (GPGPUs). GPGPUs used to rely solely on massive parallelism and shared memory to hide memory latency. However, due to the proliferation of applications with irregular memory accesses and large amounts of data, modern GPGPUs rely on caches to help mitigate the intrinsic delay incurred from memory accesses [25].

Unfortunately, GPGPU cache management and design has not evolved as rapidly as CPU caches, due to decreased cache capacity per thread and cache line lifetime (the amount of time a line goes without replacement in a cache) [23]. Because of the high throughput of GPGPUs and the inadequate performance of current GPGPU caches, there can be a very large number of misses in a very small amount of time, leading to overall performance degradations and increased energy consumption.

GPU cache management becomes even more important in a High-Performance Computing (HPC) system. 10.4% of the top 500 HPC systems utilize a GPU model from the NVIDIA Tesla family as an accelerator [24]. Thus, the HPC's overall energy efficiency and performance is highly dependent on the GPU's energy efficiency and performance. As GPGPU applications' data continue to increase, storing all the data in the shared memory becomes infeasible. Thus, the GPGPU's cache becomes more important, as it reduces the memory access time for frequently accessed data.

Most current GPGPU cache management research focus on energy reduction and performance optimization through efficient cache bypassing and thread throttling. Cache bypassing [6], [7], [13], [29] mitigates thrashing in the L1 data caches by artificially extending the cache line lifetime. Bypassing schemes allow frequently reused data blocks to remain in the cache, while data requests for low-reuse blocks are made to bypass the cache. However, as the number of bypass requests increases, on-chip resource congestion increases, eventually causing the memory pipeline to stall. Thread throttling [6], [13], [29] increases the hit rate of a system's cache(s) while decreasing potential resource congestion by dynamically or statically limiting the number of warps/threads that can be run on a given streaming multiprocessor (SM). Decreasing the number of threads increases the cache capacity per thread, which in effect increases the amount of locality in the cache. As a result, multiple GPGPU cores are not utilized to their full potential.

In this paper, we explore the potentials of leveraging *autonomic cache tuning* for GPGPU cache management. Cache tuning, otherwise known as cache optimization, uses a configurable cache [28], with total size, associativity, and line size that can be dynamically adjusted during runtime to specialize the cache's configurations to executing applications' needs. Cache tuning exploits the fact that different

applications have different configuration requirements for energy efficiency and optimal performance [9]. Thus, based on each application's execution characteristics (e.g., cache miss rates, instructions per cycle, branch mispredicts, etc.), the cache can be tuned to minimize energy consumption without degrading performance. While cache tuning has been extensively studied in CPUs [12], [20], [22], this paper is the first, to the best of our knowledge, to explore cache tuning in the context of GPGPUs. Our goal in this study is to derive insights to enable the implementation of autonomous caches in emerging GPGPUs. To this end, our contributions are as follows:

- We propose that autonomic cache tuning is a viable approach to cache management in GPGPUs, and extensively analyze the potentials for improving the energy efficiency and performance of GPGPU L1 and L2 data caches.
- We analyze autonomic cache tuning in L1 and L2 data caches to derive insight into which cache level offers the most benefits for optimal performance and energy efficiency.
- We illustrate the benefits of runtime autonomic cache tuning using a tuning heuristic that dynamically determines execution GPGPU applications' L1 data cache configurations.

Through detailed experiments using GPGPU-Sim [4], a common GPGPU architectural simulator, we show that an autonomously tuned configurable L1 data cache can reduce the average energy delay product and improve the performance (with respect to IPC) by 16.5% and 18.8%, respectively, as compared to a static non-configurable cache. Our results also reveal that tuning the L2 data cache can reduce the average EDP and improve the performance by 18.1% and 13.4%, respectively, as compared to a static L2 cache. Finally, our results reveal a strong correlation between GPGPU performance and L1 data cache reservation failures [7], and provides a solid foundation for future work enabling autonomous caches in GPGPUs.

The rest of the paper is organized as follows. In Section II we provide an overview of the related work on GPU cache management strategies, configurable caches and cache tuning. We describe our methodology, simulation environment, and experimental setup in Section III. In Section IV, we present the results and analysis generated by exhaustively exploring the cache design space to evaluate the impact of different configurations on application execution. We establish the methodology and present the results from our autonomous cache tuning experiment in Section V. Finally, in Section VI, we present our conclusions and future work.

II. Background and Related Work

Due to the poor performance of caches in GPGPUs, there has been extensive research on GPU cache management schemes. This section provides an overview of current GPU cache management schemes and background on configurable caches and cache tuning, which we leverage in this work.

A. GPU Cache Management

The main reason why caches are less effective for GPUs than for CPUs is that GPUs' high throughput is mismatched with the architecture of general purpose caches [13]. Our work represents a step in the direction of making caches more viable for GPUs, and is orthogonal to current GPU cache management techniques. We review three main research directions that currently exist for optimizing caches for GPUs.

The first involves schemes forcing low-reuse memory accesses to bypass the cache in order to not evict high-reuse cache blocks. Chen et al. [6] proposed a scheme that forces memory requests to bypass the L1 data cache based on the reuse distance theory. This scheme also dynamically throttles the number of warps to prevent occupancy of too many resources when the cache is getting bypassed frequently. Dai et al. [7] introduced a model to calculate the ideal number of requests to bypass the cache as a function of the estimated number of reservation failures. Li et al. [18] observed that the number of times cache blocks are reused is very small in certain GPGPU applications and propose a scheme that bypasses the cache if a memory request associated with a block is not going to be reused.

The second popular research area involves optimizing cache indexing to efficiently populate the cache. Kim et al. [14] studied the effects of multiple indexing techniques on performance and energy consumption within GPU caches. These indexing functions include static schemes such as XORing different bits of the instruction address and polynomial modulus mapping. Additionally, they experimented with one dynamic indexing function, which uses different bits of the instruction address to index into the cache. Kim et al. [15] proposed using polynomial modulus mapping as a means of interleaving set indexes within the cache.

A third approach optimizes cache management on a warp level in order to best account for each application's specific characteristics. Khairy et al. [13] proposed a warp throttling scheme that throttles the number of active warps based on the current misses per kilo instructions (MPKI) in order to reduce the contention seen in the L1 data cache. Zhao and Wang [29] observed that irregular memory accesses are the most detrimental aspect of an application due to memory divergence and a loss of locality. In order to combat this, they devised a scheme that is present in both the memory level and the instruction level to determine the degree of irregularity of memory requests across the threads within a warp. If those memory requests surpass a certain threshold, the entire warp is forced to bypass the L1-Data cache.

B. Configurable Caches

A configurable cache [28] is a multi-banked cache that allows the cache size to be configured to specialize the

cache's configurations to executing applications. Given a physical cache, where each cache way is implemented as a different bank, the ways can be shutdown to reduce the cache size or concatenated to configure the associativity. To configure the block size, multiple physical blocks can be fetched and concatenated to logically configure larger block sizes. This configurability can be achieved using small bit-width configuration registers, and augmenting a state-of-the-art cache to support configurability results in negligible design, power and area overheads. In addition, incorporating configurability to caches has no impact on the critical path, thus making configurable caches a viable option for low-overhead cache optimization. We direct the reader to [28] for details on the configurable cache's circuitry and design.

C. Cache Tuning

Cache tuning determines the cache configurations that best satisfy an application's execution requirements. Due to potentially large design spaces (possible combination of configurations), exhaustively exploring the full cache design space is typically unrealistic. Thus, various cache tuning heuristics and algorithms [3], [11], [22] have been developed to prune the design space, in order to reduce cache tuning overheads (energy and time spent during tuning). Zhang et al. [27] presented a heuristic that automatically tunes the cache in an embedded system to a pseudo-optimal configuration when considering energy, execution time, or a combination of the two depending on the priority of the system. Gordon-Ross et al. [10] expanded upon this heuristic and introduced the Two-level Cache Tuner (TCaT), a method to automatically tune two levels of cache in an embedded system.

Cache tuning heuristics/algorithms are usually implemented using a software or hardware tuning mechanism, known as the cache tuner. Adegbija et al. [2] developed low-overhead cache tuners for multicore systems, and analyzed different cache tuner architectures for effectively tuning caches in multicore embedded systems. The authors analyzed the energy and hardware footprint of the different architectures in order to identify which architectures resulted in minimum overheads. This work illustrated the extensibility and scalability of low-overhead cache tuning to large multicore systems, and provides a foundation for the work proposed herein.

III. Methodology

This section describes our methodology for evaluating the potentials of autonomic cache tuning in GPGPU caches. We first describe our simulation environment, and thereafter, present details of our experimental setup.

A. Simulation Environment

To evaluate the effect of varying cache configurations within a GPGPU, we used the GPGPU-Sim simulator [4] to emulate a system based on the NVIDIA Fermi GTX480 [26]. We used the GPU-WATTCH simulator [17] to collect the GPU's average power for each application. To take into account both energy and delay in the evaluations, we used the energy delay product (EDP) and instructions per cycle (IPC) as our evaluation metrics. We calculated the EDP as follows:

$EDP = system\_power * (total\_application\_cycles / system\_frequency)^2$

TABLE I: Simulated base configurations

SM config	15 SM, width = 32, 700 MHz, 48 Warps/SM, 1536 threads
Cache hashing	Linear set function
Shared memory/SM	6kB
Warp scheduling	2 GTO (Greedy-then-oldest) warp schedulers/SM
Memory model	6 GDDR5 memory controllers, FR-FCFS scheduling, 924 MHz
GDDR5 timing	tCL = 12, tRP = 12, tRC = 40, tRAS = 28, tRCD = 12, tRRD = 6

Table I depicts the static base configuration parameters used for our evaluations. We modeled the GTX480 GPU because its memory hierarchy and associated performance sufficiently represent state of the art GPUs [8] [21]. Therefore, analysis and conclusions made through experimentation on the GTX480 can easily be extended to other GPU models.

The GTX480 memory hierarchy architecture has a dedicated L1 data (L1D) cache for every SM, and a unified L2 data (L2D) cache that is split into multiple partitions used by every SM in the GPGPU. Each simulation requires a configuration file in which different system parameters can be manipulated. This includes number of sets, the associativity, the block size of both L1D and L2D caches, and the number of memory partitions for the unified L2 data cache. In conjunction with the configuration information, the actual benchmark/application is input into the simulator as well. GPGPU-Sim then collects IPC, total number of instructions, and reservation failures; these statistics are also used to calculate execution time of the program. Subsequently, the average power statistic is generated post GPGPU-Sim simulation using GPU-WATTCH. We then collated and analyzed the execution and power statistics of the different cache configurations for the benchmarks to determine the impact of cache tuning in GPGPUs.

TABLE II: Ranges of the L1D parameter values

Parameter Name	L1D Range	L1D Base Value
Number of sets	16, 32, 64	16
Associativity	1, 2, 4	4
Line size (bytes)	128, 256	128

TABLE III: Ranges of the L2D parameter values

Parameter Name	L2D Range	L2D Base Value
Number of sets	64, 128, 256	64
Associativity	8, 16, 32	16
Line size (bytes)	128, 256	128
Number of partitions	6	6

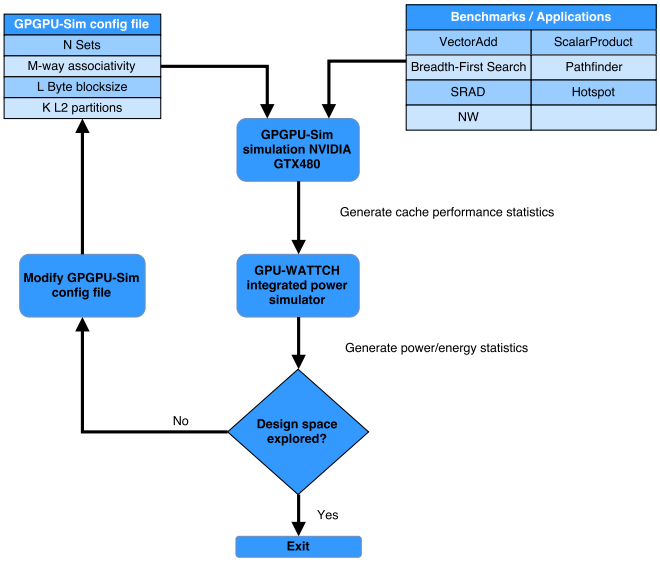


Fig. 1: Flowchart describing exhaustive search, and simulation environment and setup

B. Experimental Setup

To investigate the impact of autonomic cache tuning on GPGPU energy efficiency and performance, we first conducted an exhaustive search of the L1D and L2D cache configurations and evaluated the improvements achieved by optimal cache configurations (with respect to power consumption, IPC, and EDP) as compared to the base configuration.

Tables II and III depict the L1D and L2D cache configuration design spaces, respectively. The design space comprises 18 L1D cache configurations, with L2D cache set to the base configuration, and 18 L2D configurations, with the L1D set to the base configuration, resulting in 36 total configurations. We determined the base configurations using the most common values found in literature for the L1D ([13] [7] [29] [14]) and L2D ([13] [7] [29] [18]) caches. We assume that the caches are configured as described in Section II-B. For future work, we plan to investigate the interleaved exploration of both the L1D and L2D cache configurations, which will exponentially increase the design space.

The range for each configuration parameter is chosen to explore cache sizes larger than the base configuration, since increasing the cache size has been shown to improve performance [6]. The L1D cache associativities are smaller than the L2D cache associativities, because the L2D cache is much larger than the L1D cache. We observed that the L1D and L2D caches yield diminishing returns at associativities larger than 4 and 32, respectively.

Figure 1 depicts the sequence of steps we used to perform the exhaustive search of L1D and L2D cache configurations. We run every benchmark using GPGPU-sim in order to generate the performance statistics. GPGPU-sim then outputs a file used as an input to GPU-WATTCH in order to simulate the power statistics of the GPGPU. Then, the configuration changes to the next configuration within the configuration space. This process continues until all cache configurations are explored for every benchmark.

We tuned both the L1D and L2D caches to derive insights into which level derives the most benefit from cache tuning. Since GPGPUs typically have multiple L1D caches [8], we assume that each L1D cache is globally tuned with the same configuration, i.e., the configurations are homogeneous across the L1D caches. Thus, we assume a low-overhead hardware cache tuner that can simultaneously change the caches' configurations, without incurring significant power and area overheads. This tuner has been extensively analyzed for CPUs by prior work [2]; for future work, however, we plan to further evaluate these tuners in the context of GPGPUs. We assume a single unified L2D cache, as is the case in the GTX480 [8].

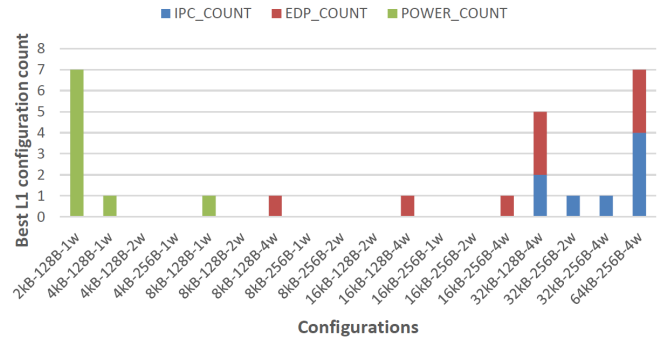
To represent a variety of GPGPU applications, we used seven benchmarks from the Rodinia 3.1 Benchmark Suite [5] and the NVIDIA Toolkit [1]. From the Rodinia 3.1 Benchmark Suite, we used BFS, HotSpot, NW, SRADv2, and Pathfinder, and from the NVIDIA Toolkit, we used VectorAdd and ScalarProd.

IV. Exhaustive Search Experimental Results

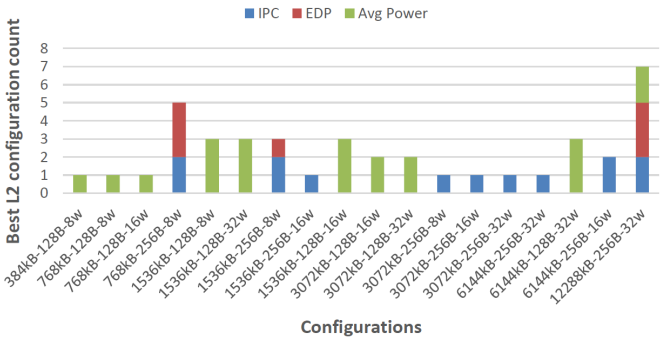
Our first set of experiments involved an exhaustive search of the L1D and L2D cache design spaces to evaluate the benefits of tuning the GPGPU's caches. The L1D caches are usually much smaller than the L2D cache, making the L1D caches more susceptible to thrashing. This observation leads us to hypothesize that the performance gain of tuning the L1D caches in the GPGPU will be greater than tuning the unified L2D cache. After our exhaustive search, we made three key observations with respect to tuning GPGPU caches. In what follows, we present each observation, and then detailed results that support the observation.

1) Different applications require different L1D and L2D cache configurations for optimal energy efficiency and performance.

To verify that cache tuning is a viable GPGPU cache optimization technique, we investigated the frequency with which different L1D and L2D cache configurations were deemed best for different evaluation metrics. Figure 2 shows a histogram displaying the frequency with which different L1D (Figure 2a) and L2D (Figure 2b) configurations were determined to be best for the different metrics.



(a) L1D best configuration distribution.



(b) L2D best configuration distribution.

Fig. 2: Distribution of best L1D (a) and L2D (b) cache configurations based on IPC, EDP, and Average Power of both. Configurations identified as CacheSize-BlockSize-Associativity

The frequency of configurations with the optimal IPC, EDP, and average power are indicated with blue, red and green bars, respectively. For the cases where two or more configurations tie for the most optimal performance within a specific benchmark, both configurations are awarded a point.

As seen in Figure 2a, when the average power is taken into account, the best configurations tend to be small, direct mapped caches. This is to be expected because cache modules with fewer sets and ways typically consume less power. When considering the IPC, larger configurations (size and associativity) tend to be the optimal configurations for the different benchmarks. Again, this observation is expected because IPC is highly related to the total throughput of the GPGPU; larger caches with large associativities typically result in fewer misses, and enable faster accesses. Finally, with EDP as the focus, the optimal cache configurations are far more spread out among the different configurations in the design space, but still favoring large caches. This can be attributed to the fact that GPGPUs are throughput-oriented devices, and small caches result in a large number reservation failures which lead to massive cache thrashing.

Figure 2b shows a more even distribution of L2D configurations that achieve the best average power and IPC. However, there were fewer configurations that achieved the best EDP across all 7 benchmarks. This implies that the performance gain from tuning the L2D cache is much smaller than tuning the L1D cache. Despite the large variation of best configurations for average power and IPC, the deviation between the statistics for each configuration is much smaller than that of the L1D cache. We also observed that EDP has a smaller distribution of best L2D cache configurations than the L1D cache. Overall, these observations indicate that tuning the L2D cache offers less optimization benefits than tuning the L1D cache; to minimize runtime overheads, the L2D cache configuration should be a single static configuration determined through extensive a priori analysis.

The key takeaway from these results is that different applications have different L1D and L2D cache configuration requirements for optimal execution. Our results show, for example, that the best configuration for minimum EDP varies significantly for the different applications. These observations, using a set of test applications, motivates us to perform analysis for a larger set of applications in future work.

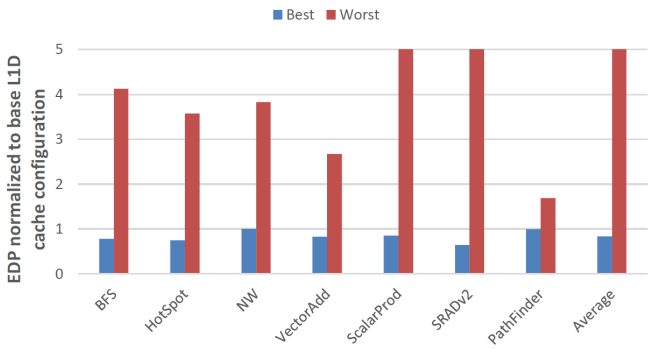
2) L1D cache tuning provides significant improvement as compared to a base static cache configuration. L2D cache tuning provides much smaller improvement.

We performed further experiments and analysis to determine which cache level offers the most improvement from cache tuning. To illustrate the potential variation of results achieved using the best, worst, and base configurations, we used exhaustive search to determine the best and worst configurations, and compared them to the base configuration.

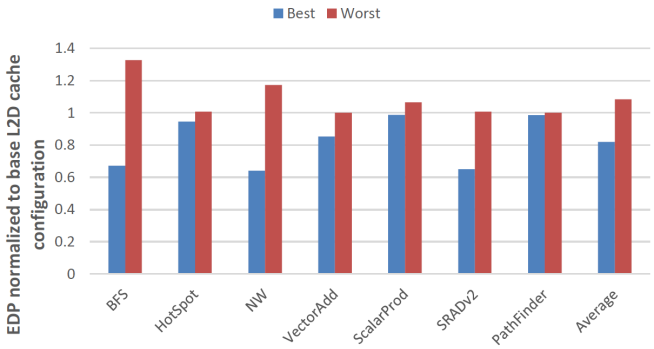
Figure 3a compares the L1D cache configurations with respect to EDP (the best and worst configurations are normalized to the base). The optimal cache configuration on average reduced the EDP by 16.5% as compared to the base cache. The EDP savings were as high as 25% and 35.5% for Hotspot and SRADv2, respectively. On the other hand, the worst cache configuration for EDP was 402% higher than the EDP of the base configuration on average. This can be attributed predominantly to two large outliers—ScalarProduct and SRADv2—whose worst configurations increased the EDP by 1110% and 613%, respectively, as compared to the base configuration.

Figure 4a compares the L2D cache configurations with respect to EDP. We observed that the optimal cache configuration reduced the average EDP by 18.1% as compared to the base configuration. The EDP savings were as high as 36% and 34.9% for NW and SRADv2, respectively. The worst configurations increased the average EDP by 8%, as compared to the base configuration. The worst configurations increased the EDP by up to 33% and 17% for BFS and NW, respectively.

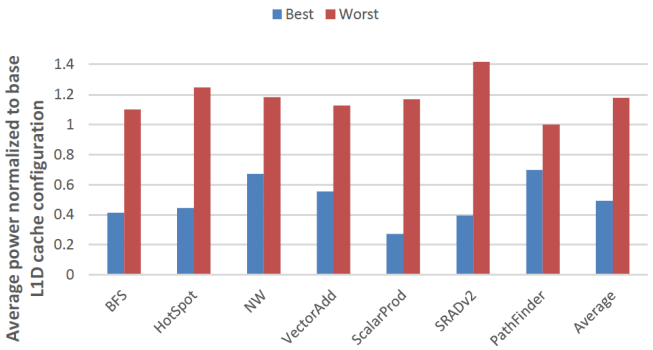
The EDP savings of the optimal L1D cache configurations were similar to those of the L2D cache configurations.



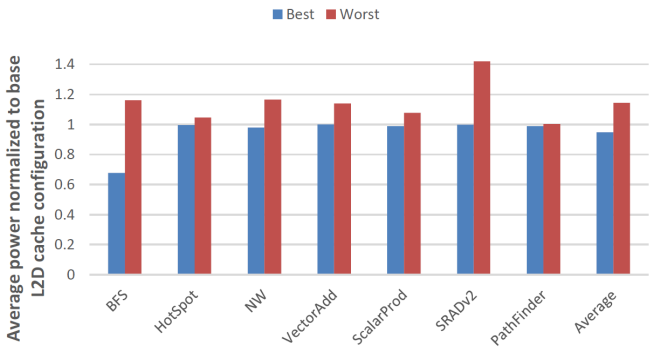
(a) EDP comparison of best, worst, and base L1D configuration.



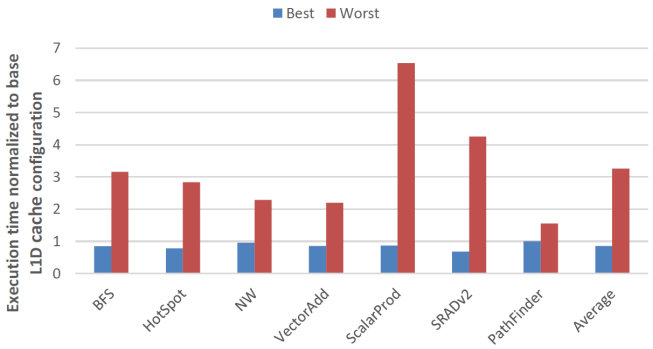
(a) EDP comparison of best, worst, and base L2D configuration.



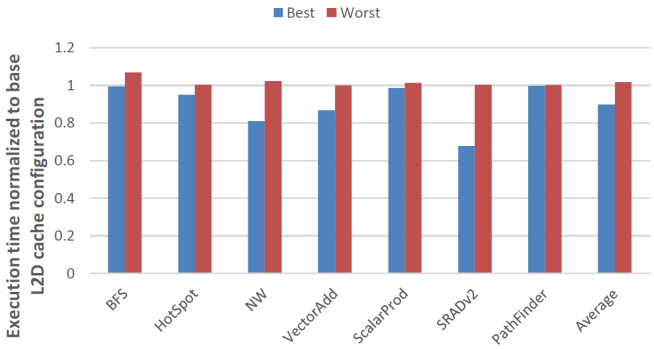
(b) Average power comparison of best, worst, and base L1D configuration.



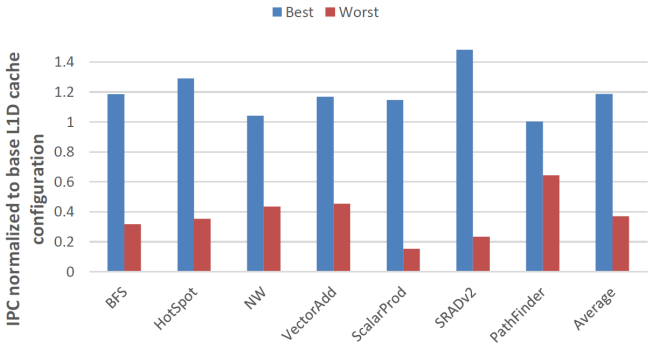
(b) Average power comparison of best, worst, and base L2D configuration.



(c) Execution time comparison of best, worst, and base L1D configuration.

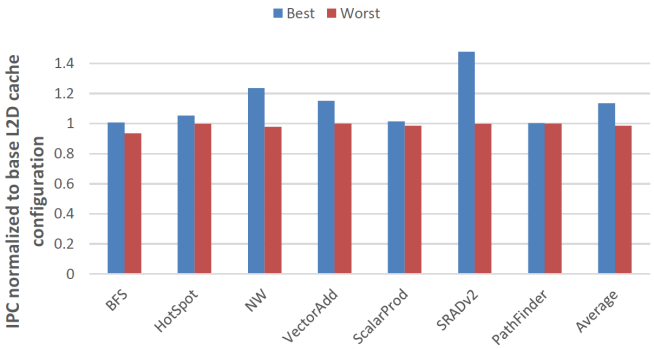


(c) Execution time comparison of best, worst, and base L2D configuration.



(d) IPC comparison of best, worst, and base L1D configuration.

Fig. 3: Cache tuning potential to improve EDP, Average Power, Execution Time, and IPC compared to base L1D cache configuration.



(d) IPC comparison of best, worst, and base L2D configuration.

Fig. 4: Cache tuning potential to improve EDP, Average Power, Execution Time, and IPC compared to base L2D cache configuration.

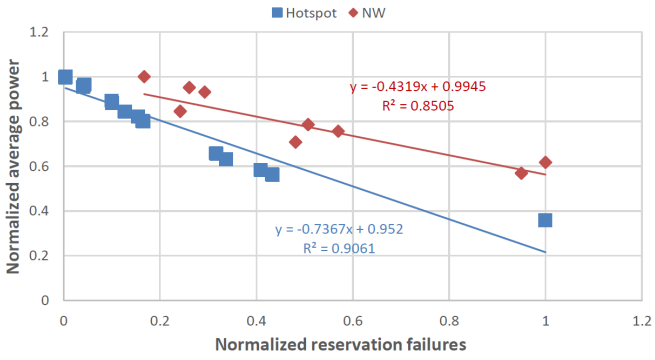
On average, tuning the optimal L1D cache configurations achieved 16.5% EDP savings, while the optimal L2D cache configurations achieved 18.1% EDP savings. However, there was a more significant difference in the EDP increase from the worst configurations. On average, the worst L1D configuration for EDP increased the EDP of the base by 402%, while the worst L2D configuration for EDP only increased the EDP by 8%. These results reveal that the GPGPU's EDP is more sensitive to changes in the L1D cache configuration than the L2D cache; a suboptimal L1D cache can result in more significant EDP degradation. Thus, optimization emphasis must be placed on the L1D cache to prevent the potential EDP increases from suboptimal L1D cache configurations.

Figure 3b demonstrates the comparison between the best, base, and worst L1D configurations when considering the average power consumption for each benchmark. On average across the benchmarks, the best configurations reduce the average power consumption by 50.7% as compared to the base configuration. For ScalarProduct and Hotspot, the best configuration reduced the power consumption by 72.9% and 55.5%, respectively. The worst configurations on average consume 17.7% more power than the base configuration.

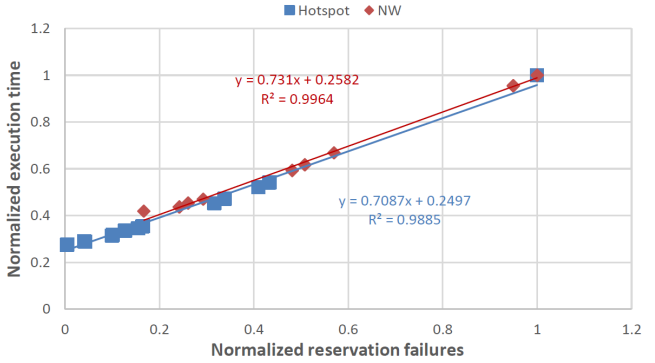
Figure 4b shows the difference in overall average power consumed by the GPGPU between the best, base, and worst L2D cache configuration for each benchmark. On average across all the benchmarks, the best configurations reduced the power consumption by 5.32% as compared to the base configuration, with power savings as high as 32.2% for BFS. On average, the worst configurations increased power consumption by 14.4%. These results show the significant disparity in the power savings obtained from tuning the L1D and L2D caches. Tuning the L2D cache reveals a smaller impact on the GPGPU's average power savings (5.32%) than changing the L1D cache configuration (50.7%). We also observed that the difference between the worst and base configuration were more similar for both L1D cache and L2D cache, with power increases of 17.7% and 14.4%, respectively.

Figure 3c compares the different L1D cache configurations with respect to execution time of the different benchmarks. Because the base configuration was relatively large, the benefits of cache tuning for execution time savings was much smaller for all the benchmarks than power and EDP. Furthermore, we observed dramatic execution time degradations with sub-optimal configurations. On average across all the benchmarks, the best configurations reduced the execution time by 14.6%, as compared to the base, while the worst configurations increased the execution time by 226%.

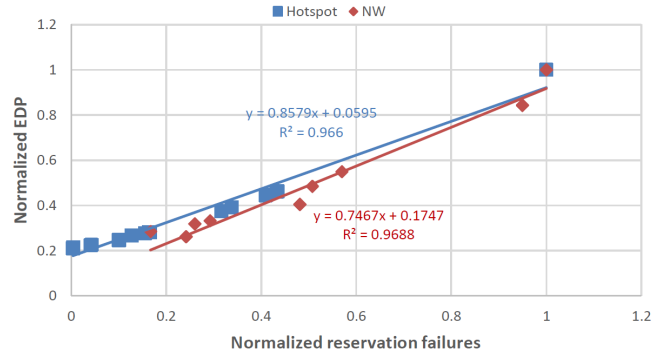
Figure 4c shows the comparison between the best, base, and wost L2D cache configurations with respect to execution time. The best L2D cache configuration reduced the average execution time by 10.3%, as compared to the base configuration, with reductions as high as 32.3% and



(a) Average power vs reservation failures



(b) Execution time vs reservation failures



(c) EDP vs reservation failures

Fig. 5: Relationship between normalized reservation failures and average power, execution time, and EDP.

19.1% for SRADv2 and NW, respectively. The worst L2D configuration increased the execution time by 1.61% with respect to the base.

These results reveal that tuning the L1D and L2D caches achieve similar execution time savings with respect to the base configuration (14.6% and 10.3%, respectively). However, we observed a significant difference in the execution time degradation of the worst L1D and L2D configurations: 226% and 1.61%, respectively.

Figure 3d compares the best and worst L1D configurations to the base, with respect to IPC. Similar to the results from average power consumption, the best configurations increase the average IPC by 18.8% as compared

to the base configuration, with increases as high as 29% and 48% for Hotspot and SRADv2, respectively. The worst configurations decreased IPC by 63.0%.

Figure 4d looks at the same IPC comparison for the L2D cache configurations. The best L2D configuration increased the average IPC by 13.4%, as compared to the base, with increases as high as 23.6% and 47.7% for NW and SRADv2, respectively. The worst L2D configuration only decreased the average IPC by 1.53% compared to the base. These results reveal similar impacts of tuning the L1D and L2D caches on IPC. However, the IPC reduction resulting from the worst configurations was more significant for the L1D cache (63%) than the L2D cache (1.53%).

Figures 3 and 4 show that the GPGPU's average IPC, power, execution time, and EDP are highly sensitive to both the L1D and L2D cache configuration. However, in general, our results reveal that the L1D cache tuning creates much larger deviations in the results than L2D cache tuning. For example, while tuning the L1D cache, the standard deviation of the IPC and average power across all configurations was 63.46 and 15.78, respectively. While tuning the L2D cache data, the average standard deviation of IPC and average power across all configurations was 6.936 and 3.692, respectively. These results show that while both L1D and L2D cache tuning can improve a GPGPU's efficiency, tuning the L1D cache offers more promise and warrants more optimization efforts than tuning the L2D cache.

3) L1D cache reservation failures can be used for performance prediction during cache tuning.

The GPU handles misses with the Miss Status/Information Holding Registers (MSHR) [16]. There are multiple MSHRs per Streaming Multiprocessor (SM); each MSHR can track one pending memory request. However, if there are many misses in a small period of time, a SM can run out of resources, which include available MSHRs as well as miss queue entries and available cache lines. This is referred to as a reservation failure, and it requires stalling the entire memory pipeline until sufficient resources are available to handle additional cache misses [7].

Our results indicate that as L1D reservation failures increase, the GPGPU's overall average power consumption decreases. We show this trend in Figure 5a using two candidate benchmarks, NW and Hotspot, with respective correlation coefficients (R) of -0.9222 and -0.9519. On average, all benchmarks have a coefficient of determination (R^2) of 0.90169, indicating a strong correlation between the power consumption and reservation failures for all applications. This trend occurs because as the cache size decreases the reservation failures increase, as there are fewer resources available and the cache is unable to keep up with the demands from the GPGPU [13]. GPGPUs are designed to have high throughput, therefore, smaller caches have more problems keeping all the necessary data in the cache at a given time. This causes a large amount of cache misses, resulting in poorer performance and subsequently more reservation failures.

Figure 5b shows a linear correlation between execution time and reservation failures. The correlation coefficients (R) of NW and Hotspot are 0.9982 and 0.9942. We observed a similar trend for all the benchmarks: on average, the benchmarks have coefficient of determination (R^2) of 0.9907 (figures omitted for brevity). We also observed similar trends for the EDP, as shown in Figure 5c. Given these observations, we conjecture that the reservation failures can be used for predicting L1D configurations during runtime. We intend to exploit these correlations for cache tuning in future work.

We found that, unlike the L1D cache, there was no correlation between the L2D cache and reservation failures. This can be attributed to the fact that there are many more reservation failures for the L1D cache than the L2D cache. The average number of L1D reservation failures across all benchmarks and all configurations was 529.22x the L2D reservation failures. There were more L1D cache reservation failures because there are many more L1D caches than L2D caches—15 L1D caches in the GTX480 vs one unified L2D cache—and the L2D caches are much larger than the L1D caches. Thus, the L2D cache is able to maintain the locality of several executing threads. For the L2D cache, the coefficients of determination (R^2) between the reservation failures and average power, execution time, and EDP were 0.32231, 0.47052, and 0.49035, respectively (figures omitted for brevity).

V. Autonomic Cache Tuning Analysis

In addition to conducting an exhaustive search of all possible L1D and L2D cache configurations, we used a simple heuristic, similar to the one proposed in [27], to dynamically predict the optimal L1D cache configuration for executing applications. We assume that the heuristic is implemented by a low-overhead hardware cache tuner [2]. This tuner has been extensively analyzed to show that it contributes minimal power and area overhead to a resource-constrained embedded system. In addition, the accrued tuning stall cycles do not result in any significant runtime tuning overhead.

To take into account both energy and execution time, the heuristic uses the EDP as the optimization metric; however, the heuristic can easily be modified to optimize for energy or execution time separately. The heuristic operates as follows:

1) Begin with the smallest cache: 16 sets, 128 byte line size, direct mapped. Increase the number of sets to 32 and keep the other two parameters constant. If the change in number of sets decreases the EDP, increase the number of sets to 64. Pick the number of sets corresponding to the lowest overall EDP.

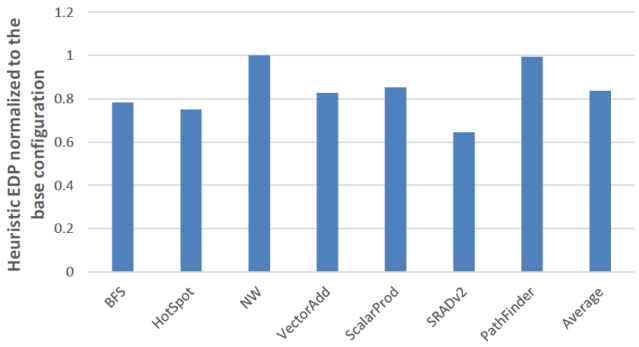


Fig. 6: Heuristic EDP normalized to the base L1D cache configuration.

2) With the number of sets and associativity constant, increase the line size from 128 to 256 bytes. If the increase in line size decreases EDP, choose 256 bytes as the line size. If increasing the line size increases EDP, choose 128 bytes as the line size.
3) With the number of sets and line size constant, increase the associativity from direct mapped to 2-way set associative. If this change decreases the EDP, increase the associativity from 2-way to 4-way set associative. Pick the associativity corresponding to the lowest overall EDP.

Figure 6 shows that for every benchmark, the heuristic determined the optimal configuration for EDP. Similar to exhaustive search, the heuristic achieved average EDP savings of 16.5%, as compared to the base cache configuration, with savings as high as 35.6% for SRADv2. The heuristic achieved these EDP savings while exploring 31.75% of the design space. We expect this percentage to reduce even further with a larger design space.

VI. Conclusions

In this paper, we explored autonomic cache tuning for optimizing the L1 and L2 data caches in general purpose graphics processing units (GPGPU). Cache tuning adapts the system cache configurations to executing applications' resource requirements in order to reduce energy or improve performance. We showed that different GPGPU applications require different runtime L1 and L2 data cache configurations for optimal execution. Thus, using extensive experiments, we showed that L1 data cache tuning can reduce average power and EDP by 50.7% and 16.5%, respectively, and increase the average IPC by 18.8%, as compared to a static L1 data cache. We also showed that, in general, the L1 data cache offers more promise for GPGPU optimization than the L2 data cache. Our results also revealed that the reservation failures may be used to predict when to switch to a new cache configuration and what that configuration may be. Finally, we implemented a simple heuristic to autonomously tune the L1D cache, showing optimal configuration predictions in all applications while significantly reducing the design space.

For future work, we intend to explore an interlaced tuning of the L1 and L2 caches. This interlaced tuning will significantly increase the design space, and potentially lead to higher optimization potential. Given a larger design space, we also intend to explore techniques for optimizing the cache tuning heuristic to determine optimal configurations while exploring a small subset of the design space. We also intend to design a prediction scheme, leveraging reservation failures, to predict the best cache configurations for application phases in a GPGPU. Also, we plan to extensively evaluate and explore additional techniques for implementing configurable caches and cache tuning in GPGPUs, while accruing minimal runtime overheads.

References

[1] Nvidia cuda toolkit. https://developer.nvidia.com/cuda-toolkit Accessed: February 2017.
[2] T. Adegbija, A. Gordon-Ross, and M. Rawlins. Analysis of cache tuner architectural layouts for multicore embedded systems. In *Performance Computing and Communications Conference (IPCCC), 2014 IEEE International*, pages 1–8. IEEE, 2014.
[3] M. H. Alsafrjalani, A. Gordon-Ross, and P. Viana. Minimum effort design space subsetting for configurable caches. In *Embedded and Ubiquitous Computing (EUC), 2014 12th IEEE International Conference on*, pages 65–72. IEEE, 2014.
[4] A. Bakhoda, G. L. Yuan, W. W. Fung, H. Wong, and T. M. Aamodt. Analyzing cuda workloads using a detailed gpu simulator. In *Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on*, pages 163–174. IEEE, 2009.
[5] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron. Rodinia: A benchmark suite for heterogeneous computing. In *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*, pages 44–54. Ieee, 2009.
[6] X. Chen, L.-W. Chang, C. I. Rodrigues, J. Lv, Z. Wang, and W.-M. Hwu. Adaptive cache management for energy-efficient gpu computing. In *Proceedings of the 47th annual IEEE/ACM international symposium on microarchitecture*, pages 343–355. IEEE Computer Society, 2014.
[7] H. Dai, C. Li, H. Zhou, S. Gupta, C. Kartsaklis, and M. Mantor. A model-driven approach to warp/thread-block level gpu cache bypassing. In *Design Automation Conference (DAC), 2016 53nd ACM/EDAC/IEEE*, pages 1–6. IEEE, 2016.
[8] GeForce. *GTX480 Specifications*. http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-480/specifications Accessed: May 2017.
[9] A. Gordon-Ross and F. Vahid. A self-tuning configurable cache. In *Proceedings of the 44th annual Design Automation Conference*, pages 234–237. ACM, 2007.
[10] A. Gordon-Ross, F. Vahid, and N. Dutt. Automatic tuning of two-level caches to embedded applications. In *Proceedings of the conference on Design, automation and test in Europe-Volume 1*, page 10208. IEEE Computer Society, 2004.
[11] H. Hajimiri and P. Mishra. Intra-task dynamic cache reconfiguration. In *VLSI Design (VLSID), 2012 25th International Conference on*, pages 430–435. IEEE, 2012.
[12] H. Hajimiri, P. Mishra, and S. Bhunia. Dynamic cache tuning for efficient memory based computing in multicore architectures. In *VLSI Design and 2013 12th International Conference on Embedded Systems (VLSID), 2013 26th International Conference on*, pages 49–54. IEEE, 2013.
[13] M. Khairy, M. Zahran, and A. G. Wassal. Efficient utilization of gpgpu cache hierarchy. In *Proceedings of the 8th Workshop on General Purpose Processing using GPUs*, pages 36–47. ACM, 2015.
[14] K. Y. Kim and W. Baek. Quantifying the performance and energy efficiency of advanced cache indexing for gpgpu computing. *Microprocessors and Microsystems*, 43:81–94, 2016.

[15] K. Y. Kim, J. Park, and W. Baek. Iacm: Integrated adaptive cache management for high-performance and energy-efficient gpgpu computing. In *Computer Design (ICCD), 2016 IEEE 34th International Conference on*, pages 380–383. IEEE, 2016.

[16] A. Lashgar, E. Salehi, and A. Baniasadi. Understanding outstanding memory request handling resources in gpgpus. In *proceedings of The Sixth International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies (HEART'2015)*, 2015.

[17] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M. Aamodt, and V. J. Reddi. Gpuwattch: enabling energy optimizations in gpgpus. In *ACM SIGARCH Computer Architecture News*, volume 41, pages 487–498. ACM, 2013.

[18] C. Li, S. L. Song, H. Dai, A. Sidelnik, S. K. S. Hari, and H. Zhou. Locality-driven dynamic gpu cache bypassing. In *Proceedings of the 29th ACM on International Conference on Supercomputing*, pages 67–77. ACM, 2015.

[19] A. Malik, B. Moyer, and D. Cermak. A low power unified cache architecture providing power and performance flexibility. In *Low Power Electronics and Design, 2000. ISLPED'00. Proceedings of the 2000 International Symposium on*, pages 241–243. IEEE, 2000.

[20] S. Mittal. A survey of architectural techniques for improving cache power efficiency. *Sustainable Computing: Informatics and Systems*, 4:33–43, 2014.

[21] NVIDIA. *Tesla GPU Accelerators*. http://www.nvidia.com/content/tesla/pdf/nvidia-tesla-kepler-family-datasheet.pdf Accessed: May 2017.

[22] M. Rawlins and A. Gordon-Ross. An application classification guided cache tuning heuristic for multi-core architectures. In *Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific*, pages 23–28. IEEE, 2012.

[23] T. G. Rogers, M. O'Connor, and T. M. Aamodt. Cache-conscious wavefront scheduling. In *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 72–83. IEEE Computer Society, 2012.

[24] E. Strohmaier, H. Meuer, J. Dongarra, and H. Simon. The top500 list and progress in high-performance computing. In *Computer*, pages 42–49. IEEE Computer Society, 2015.

[25] C. M. Wittenbrink, E. Kilgariff, and A. Prabhu. Fermi gf100 gpu architecture. *IEEE Micro*, 31(2):50–59, 2011.

[26] C. M. Wittenbrink, E. Kilgariff, and A. Prabhu. Fermi gf100 gpu architecture. *IEEE Micro*, 31(2):50–59, 2011.

[27] C. Zhang and F. Vahid. Cache configuration exploration on prototyping platforms. In *Rapid Systems Prototyping, 2003. Proceedings. 14th IEEE International Workshop on*, pages 164–170. IEEE, 2003.

[28] C. Zhang, F. Vahid, and W. Najjar. A highly configurable cache architecture for embedded systems. In *Computer Architecture, 2003. Proceedings. 30th Annual International Symposium on*, pages 136–146. IEEE, 2003.

[29] C. Zhao, F. Wang, Z. Lin, H. Zhou, and N. Zheng. Selectively gpu cache bypassing for un-coalesced loads. In *Parallel and Distributed Systems (ICPADS), 2016 IEEE 22nd International Conference on*, pages 908–915. IEEE, 2016.