A Locality-Aware Memory Hierarchy for Energy-Efficient GPU Architectures

Minsoo Rhu Michael Sullivan Jingwen Leng Mattan Erez
{minsoo.rhu, mbsullivan, jingwen, mattan.erez}@utexas.edu
Department of Electrical and Computer Engineering
University of Texas at Austin
Austin, Texas 78712-1684

ABSTRACT

As GPU's compute capabilities grow, their memory hierarchy increasingly becomes a bottleneck. Current GPU memory hierarchies use coarse-grained memory accesses to exploit spatial locality, maximize peak bandwidth, simplify control, and reduce cache meta-data storage. These coarse-grained memory accesses, however, are a poor match for emerging GPU applications with irregular control flow and memory access patterns. Meanwhile, the massive multi-threading of GPUs and the simplicity of their cache hierarchies make CPU-specific memory system enhancements ineffective for improving the performance of irregular GPU applications. We design and evaluate a locality-aware memory hierarchy for throughput processors, such as GPUs. Our proposed design retains the advantages of coarse-grained accesses for spatially and temporally local programs while permitting selective fine-grained access to memory. By adaptively adjusting the access granularity, memory bandwidth and energy are reduced for data with low spatial/temporal locality without wasting control overheads or prefetching potential for data with high spatial locality. As such, our locality-aware memory hierarchy improves GPU performance, energy-efficiency, and memory throughput for a large range of applications.

Categories and Subject Descriptors

C.1.2 [**Multiple Data Stream Architectures**]: Single-instruction-stream, multiple-data-stream processors (SIMD)

General Terms

Design, Experimentation, Performance

Keywords

GPU, SIMD, SIMT, Fine-Grained Memory Access, Irregular Memory Access Patterns, Adaptive Granularity Memory

1. INTRODUCTION

General purpose computation with graphics processing units (GPUs) has become increasingly popular thanks to the high compute throughput and peak memory bandwidth of these devices. A coarse-grained (CG) memory hierarchy[1] enables GPUs to exploit programs with high spatial locality, increasing peak memory bandwidth and decreasing control overheads. Regularly structured, compute-intensive applications can readily utilize the high peak memory bandwidth and ample computational resources of GPUs to great effect. However, not all applications can be re-factored to exhibit regular control flow and memory access patterns, and many emerging GPU applications suffer from inefficient utilization of off-chip bandwidth and compute resources [1, 2, 3]. Recent proposals have primarily focused on overcoming irregularity by improving device utilization and latency tolerance [4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15], but the memory bandwidth bottleneck still remains as a significant issue in future throughput computing [16].

Coarse-grained memory accesses waste off-chip bandwidth and limit the energy-efficiency of current GPUs for irregular applications by over-fetching unnecessary data. Figure 1 shows the number of 32-byte sectors (out of each 128-byte cache block) that are actually referenced over the lifetime of L1/L2 cache blocks. Regularly structured programs with high spatial and temporal locality use most or all of the sectors within each cache block, effectively utilizing the CG memory accesses of the current GPU memory hierarchy. As we study in this paper, however, the massively multithreaded nature of GPUs allows little cache capacity per thread, resulting in high cache miss rates and reducing the amount of temporal locality that can be exploited for certain applications. Such behavior (combined with the CG-only memory hierarchy) significantly over-fetches off-chip data for irregular applications, wasting memory bandwidth, on-chip storage, and DRAM power.

This paper presents the first memory hierarchy design that can efficiently handle irregular memory access patterns and scatter-gather programs in modern GPU architectures. We design a reactive and efficient memory system that is *locality-aware* such that it can cater to the behavior of irregular GPU programs. Prior work has used the dynamic estimation of *spatial* data locality for selective fine-grained (FG) memory accesses in control-intensive, general-purpose CMP environments [17, 18]. We show that such approaches, while successful for CPUs, fall short for massively multithreaded throughput-oriented GPUs because many emerging GPU applications with irregular control/memory accesses exhibit low temporal locality and caching efficiency. To this end,

[1]A coarse-grained memory hierarchy is currently assumed to be part of the baseline GPU design in the academic community (Section 2.2).

MICRO'46 December 7–11, 2013, Davis, CA, USA.

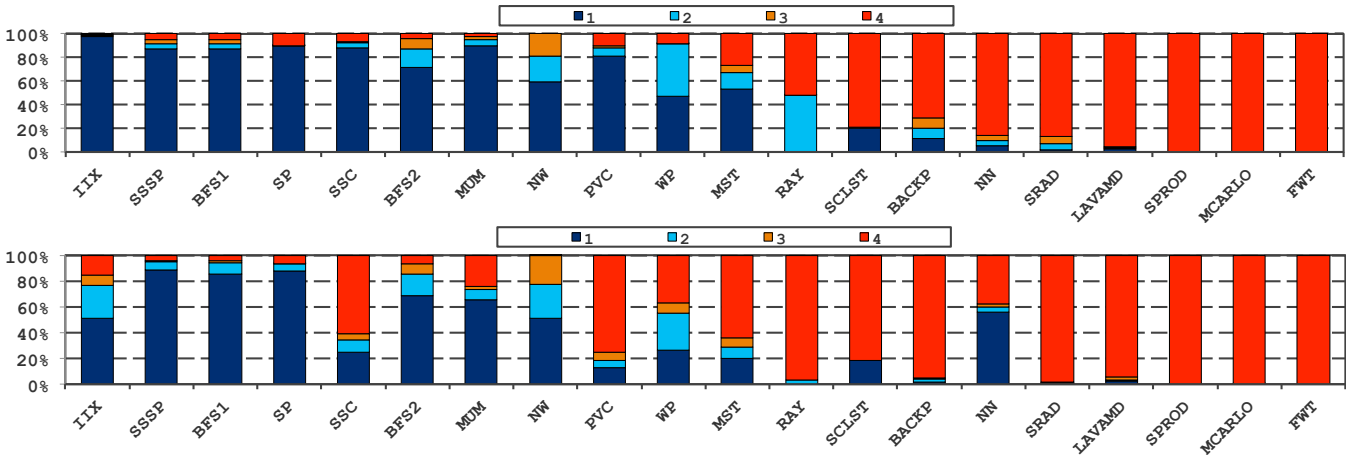


Figure 1: The number of sectors referenced in L1 (top) and L2 (bottom) cache blocks using a *CG*-only memory hierarchy. Each 128-byte cache block is logically divided into 4 32-byte sectors. While some applications (e.g., SPROD, MCARLO, FWT, etc) utilize most of the fetched data, thereby maximizing the benefits of *CG* memory accesses, others (e.g., IIX, SSSP, etc) over-fetch memory sectors, inefficiently utilizing the *CG*-only memory hierarchy.

our *locality-aware memory hierarchy* (*LAMAR*) provides the ability to tune the memory access granularity for GPUs. We first discuss the possibility of statically determining *CG*/*FG* decisions (guided by a profiler/autotuner) such that the memory access granularity best suits application characteristics. We then develop a scalable, low-cost hardware predictor that adaptively adjusts the memory access granularity without programmer or runtime system intervention. *LAMAR* maintains the advantages of *CG* accesses for programs with high spatial and temporal locality, while selective *FG* accesses reduce over-fetching and enable more efficient off-chip bandwidth utilization. By using multiple granularity memory accesses in a manner appropriate for the GPU memory hierarchy, *LAMAR* improves the efficiency of a wide range of GPU applications, significantly improving memory bandwidth, energy-efficiency, and overall performance. Also, the implementation of *LAMAR* is kept transparent to the user and without major changes to the underlying microarchitecture, easing its adoption into future GPU systems. To summarize our most important contributions:

- This paper provides a thorough, quantitative analysis of the primary inefficiencies in the *CG* memory hierarchy. We analyze the root cause behind the inefficient caching and low locality of irregular applications and make a case for a locality-aware GPU memory hierarchy.
- Unlike previous literature that aims to improve the compute resource utilization or latency tolerance, we propose the first design that resolves the under-utilization of memory bandwidth in GPU architectures.
- We demonstrate that *LAMAR* does not harm the performance or energy-efficiency of the *CG*-only baseline for conventional programs with good caching behavior and regular control/memory accesses, while significantly improving the energy-efficiency of irregular applications.

2. BACKGROUND

2.1 CUDA Programming Model

Current GPUs [19, 20, 21] consist of multiple shader cores[2], with each core containing a number of SIMD lanes for vector instructions. In NVIDIA's CUDA programming model, scalar threads are grouped into *thread blocks* that execute a single program (or *kernel*). To facilitate the execution of scalar threads in the SIMD pipeline, the thread scheduler coordinates the issue/execution of threads at a *warp* granularity; currently, a warp is defined as a group of 32 threads executing in lockstep. Because current GPUs are built around the *single-instruction multiple-thread* (SIMT) model [22, 23], each SIMD lane can execute its own logical thread with hardware support for independent branching and load/store instructions. This native support for diverging scalar threads allows memory accesses to exhibit fine-grained *scatter-gather* characteristics, as memory addresses are determined at a per-thread granularity. This means a warp can generate up to 32 independent memory transactions. To reduce the control overheads of a memory operation, GPUs contain a *memory-coalescing unit* that agglomerates the memory requests of active threads within each warp; whose requesting data size can ranges from 32 to 128-byte.

While *FG* memory accesses are allowed by the programming model, current GPU memory system is ill-suited for efficient execution of such workloads. Rather, the GPU memory system is optimized for *CG* accesses, as described below.

2.2 Coarse-Grained GPU Memory Systems

GPU manufacturers do not reveal deep microarchitectural details of their memory systems, so previous literature [8, 10, 11, 12, 15] assumes a baseline GPU memory system optimized for *CG* memory accesses (Figure 2). This is based on the fact that each memory-channel of Fermi (GF110, [19]), Kepler (GK110, [20]), and SouthernIslands [21] are 64-bits, making a 64-byte minimum-access granularity (8-bursts, 64-bits per burst) with GDDR5 chips [24]. Following memory coalescing, accordingly, the off-chip memory system of GPUs presents a uniform interface to the processor with a large minimum access granularity of cache block size, as shown in Figure 2.

2.3 Fine-Grained Data Management

Coarse-grained accesses can be useful, as they reduce miss rates and amortize control costs for spatially and temporally local requests. Emerging applications, however, exhibit dynamic and heterogeneous amounts of spatial locality, and the massively-multithreaded GPU architecture limits the temporal locality that can be exploited. In the absence of

[2]Each GPU shader core is referred to as a streaming multiprocessor (SM) in NVIDIA's CUDA terminology; we share this terminology throughout the rest of this paper.

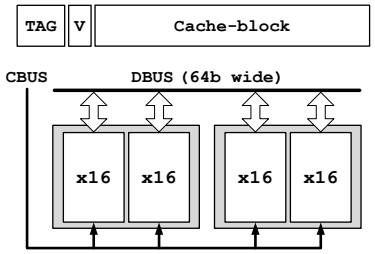


Figure 2: The baseline memory hierarchy using a *CG*-only memory system with a 64-byte (64b × 8-bursts) minimum access granularity per channel. CBUS and DBUS represent the command and data bus, respectively.

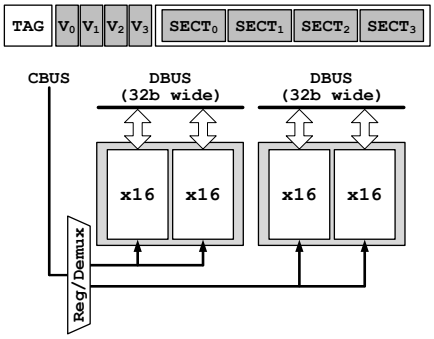


Figure 3: A memory system with two sub-ranks, providing a 32-byte (32b × 8-bursts) minimum access granularity. When used with a sector cache, the size of each sector is 32-bytes.

high locality, *FG* memory accesses avoid unnecessary data transfers, save power, and improve system performance. Section 2.3.1 and Section 2.3.2 introduce key microarchitectural structures that are adopted by this work to enable the *FG* management and storage of data.

2.3.1 Sub-ranked Memory

A conventional GPU memory system uses multiple DRAM chips organized in a rank to provide coarse-grained accesses to memory. In order to exploit the benefits of *FG* accesses for irregular workloads, *LAMAR* must reduce the minimum access granularity to off-chip memory. To do so, *LAMAR* leverages a *sub-ranked* memory system to non-intrusively allow fine-grained memory requests. The sub-ranked memory system adopted by *LAMAR* is inspired by many prior works, including HP's MC-DIMM (multi-core dual in-line memory module) [25, 26], Rambus's micro-threading [27] and threaded memory module [28], the mini-rank memory system [29], and Convey's S/G DIMM (scatter/gather dual in-line memory module) [30]. In a sub-ranked DIMM, peripheral circuitry is used to divert memory command signals to a sub-rank of DRAM chips without changing the DRAM structure itself. Figure 3 shows the sub-ranked memory system used for *LAMAR*, which provides a minimum access granularity of 32-bytes, equivalent to the smallest data request generated by an SM [22]. Because *LAMAR* provides an adaptive access granularity to suit program needs, *CG* memory requests are also allowed—with sub-ranking disabled, *CG* memory requests proceed as normal.

2.3.2 Fine-Grained Cache Architecture

Fine-grained memory accesses require some cache changes to maintain *FG* information in the on-chip memory hierarchy.

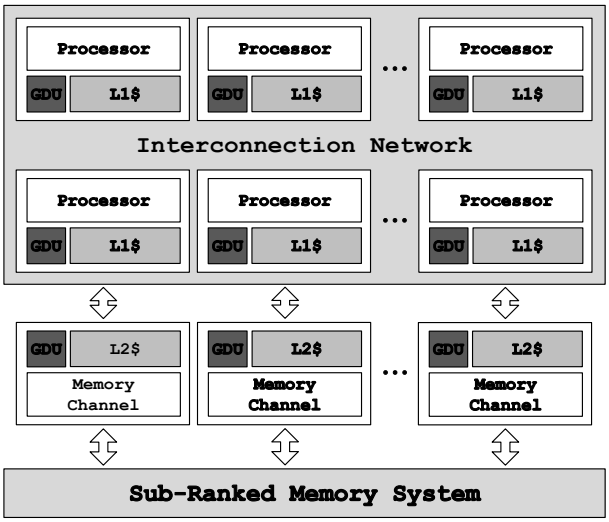


Figure 4: The proposed *LAMAR* GPU architecture. Each on-chip cache is sectored and is augmented with a GDU; off-chip memory is sub-ranked in order to allow fine-grained accesses.

This work utilizes a simple sector cache [31] to enable the on-chip management and storage of *FG* data. The sector cache partitions each cache block into sectors, each with its own validity meta-data; this allows for data to be managed at a granularity finer than a cache block. Figure 3 illustrates how data are partitioned and stored into a sectored cache block. The sector cache used for *LAMAR* partitions each cache block into 4 32-byte sectors.

2.4 Related Works

Prior work uses programmer-annotated [17, 32] or dynamically estimated [18] spatial locality data for multi-granularity memory accesses in a control-intensive, general-purpose CMP environment. This study demonstrates that these adaptive-granularity memory systems fall short for throughput processors, as they lack consideration for massive multithreading.

As was previously mentioned, *CG*-only memory systems inefficiently utilize off-chip bandwidth in the presence of program irregularity. As such, high-end vector processors (such as Cray's Black Widow [33]) sometimes use a *FG*-only memory system approach to good effect. However, such systems squander the benefits of *CG* accesses for programs with ample locality. It is unlikely that a *FG*-only approach would be competitive in the GPU market, where the performance of regular graphics workloads is of paramount importance.

3. LOCALITY-AWARE MEMORY SYSTEM

This work is motivated by the dual observations that many GPU applications demonstrate highly dynamic and heterogeneous amounts of spatial locality, and that the low per-thread cache capacity of throughput processors limits the amount of temporal locality that can be exploited by running programs. With a *CG*-only memory system, these factors conspire to squander off-chip memory bandwidth and energy by over-fetching unnecessary data (Figure 1). Established techniques estimate and exploit the *spatial* locality of cache blocks in multi-core CMPs to deal with a similar phenomenon [17, 18, 34, 35]. However, as we demonstrate, these prior techniques do not provide robust benefits for all GPU applications because they do not consider the *temporal* locality of data. We make

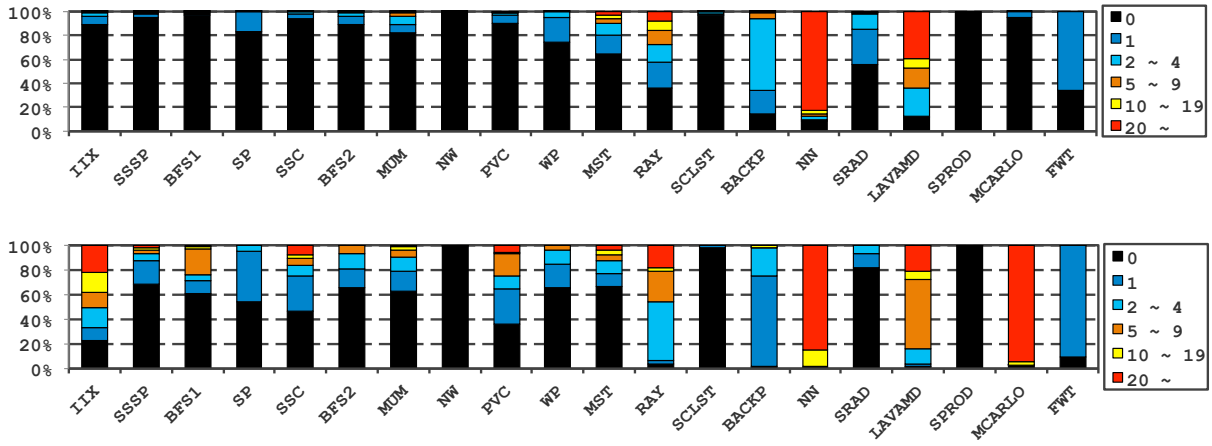


Figure 5: The distribution of repeated accesses to cache blocks in the L1 (top) and L2 (bottom) caches (using a CG-only memory hierarchy).

the case for a *locality-aware memory hierarchy* (*LAMAR*) that provides robust, effective multi-granularity accesses for throughput processing.

3.1 High-Level Overview of LAMAR

LAMAR uses a sector cache (both L1 and L2) and a sub-ranked memory system[3] in order to demonstrate the full benefits of our proposed scheme (Figure 4). The width of each sector, as well as the minimum access granularity of the sub-ranked memory system, is equivalent to the smallest data request size generated by the memory-coalescing unit within the SM, which in current generation of GPUs is 32-bytes (Section 2.1)[4]. Each cache is augmented with a *granularity decision unit* (GDU) that determines the access granularity of each cache miss. In the baseline *CG*-only memory system, all cache misses are requested at a cache block granularity, whereas *LAMAR* leverages the GDU to determine which access granularity best suits the application.

3.2 Statically-Driven GDU

LAMAR provides the programmer the option to tune the access granularity by statically designating whether to fetch all program data at a coarse or fine granularity. Such a decision may be guided by profilers/autotuners and is sent to the runtime system (e.g., through compiler options, APIs, etc.) to update each GDU. Skilled programmers can therefore configure the GDU as appropriate to the application's needs, achieving optimal bandwidth utilization and energy-efficiency. As detailed in Section 5, we find the *average number of sectors referenced within a cache block* (Table 3, Avg_{sec}) to be a good metric for characterizing the program access granularity.

3.3 Dynamically Adaptive GDU

Despite the advantages of a statically-driven GDU, identifying and specifying the optimal access granularity requires both extra effort from the programmer and system support. To this end, we describe a hardware-only mechanism that dynamically derives the optimal access granularity at runtime, achieving comparable benefits of the statically-driven GDU in a robust manner across all studied applications.

3.3.1 Spatial-Pattern Predictor

Previous work exploits adaptive granularity memory accesses in a multi-core CMP system [18] using a spatial-pattern predictor [34, 35] (SPP) in place of the GDU. Spatial pattern prediction uses a pattern history table (PHT) to collect and predict likely-to-be-used sectors upon a cache miss. Each cache block in an SPP-based system is augmented with a set of *used bits* that designate whether a given sector has been referenced or not. When a cache block is evicted, the corresponding used bit information is committed to the PHT. Future misses to each block query the PHT to determine which sectors are likely to be referenced in the future, allowing targeted sector fetches. Details of the microarchitectural aspects of the SPP can be found in [18, 35].

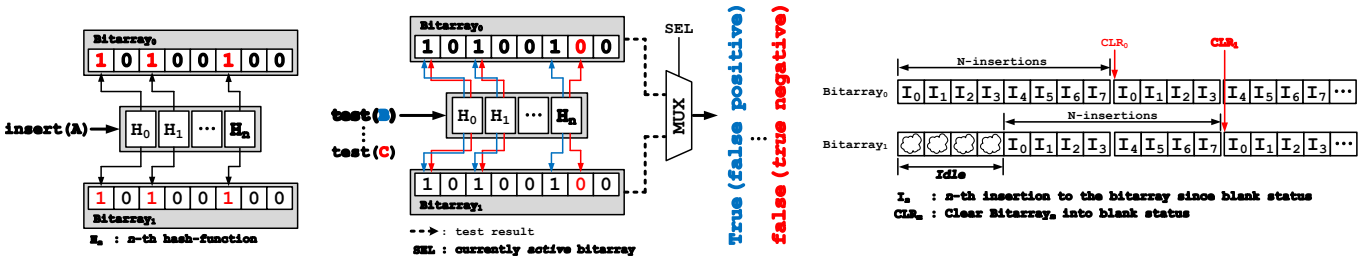
3.3.2 Pitfalls of SPP-based GDU in GPUs

While spatial pattern prediction has been effectively employed in multi-core CMP research, we explain in this section why the SPP does not perform well in the massively multithreaded GPU environment. As pointed out in previous literature [10, 36], many GPU applications do not cache well and suffer from high cache miss rates and low block reuse. Such low caching efficiency occurs both due to streaming data accesses and also because the threads contend for cache resources and constrain the effective on-chip storage available to each thread. While massive multithreading enables GPUs to be highly latency tolerant, it comes at the cost of poor cache performance, which (combined with the CG-only memory system) wastes memory bandwidth and can limit system energy-efficiency.

The SPP is not as effective for multi-granularity access in GPUs as it is in CMPs, because the high cache turnover rate and low cache block reuse of GPUs significantly lowers the temporal locality that can be exploited by the on-chip memory hierarchy. Figure 5 shows the distribution of *repeated accesses* across all cache blocks in the baseline memory system. 12 of the 20 benchmarks suffer from poor cache block reuse due to low temporal locality and high on-chip storage contention, resulting in more than 50% of the L2 cache blocks never being reused before eviction. While the SPP accurately estimates the spatial locality of data, it is not robust in the presence of low temporal locality and poor cache performance

[3]We also discuss the implications of *LAMAR* with minimum changes to the GPU architecture by only using a sectored L1/L2 cache *without* a sub-ranked memory system, the result of which is detailed in Section 5.5.

[4]Enabling a minimum access granularity smaller than 32-bytes requires restructuring the memory-coalescing unit in the SM. In this work, we leverage the current SM architecture as-is to demonstrate the benefits of *LAMAR* while minimizing the changes to the current GPU architecture.



(a) *Inserting* an element. (b) *Testing* whether an element is within the set. (c) Operation of a dual-bitarray bloom filter.

Figure 6: A microarchitectural overview of the proposed dual-bitarray bloom filter. Note that A, B and C in (a, b) are distinct values. All insertions into the bloom filter are applied to *both* bitarrays (a), except during the initial idle period of $Bitarray_1$ (c). To minimize the false positive rate, each bitarray is *cleared* after N insertions at which point the *active* bitarray (determined by SEL in (b)) is swapped. This dual-bitarray microarchitecture allows the bloom filter to retain at least a $(\frac{N}{2})$ insertion history from the newly active bitarray, avoiding periods where it contains zero history.

as we quantitatively demonstrate in Section 5.1.

3.4 Bi-Modal Granularity Predictor

In general, we observe that the SPP fails to provide robust prediction quality and high energy-efficiency to a wide range of GPU applications. While a more sophisticated prediction algorithm could potentially enhance the effectiveness of SPP, the complexity and high area overhead of a modified SPP will not scale to a many-core environment. We propose a simple, low cost *bi-modal granularity predictor* (*BGP*) that is much more suitable for throughput-oriented architectures.

3.4.1 Key Idea and Observation

The main inefficiency of spatial pattern prediction is that the spatial locality information tracked by the PHT is useless for cache blocks with low temporal locality. With this in mind, our proposed *BGP* microarchitecture is structured such that it estimates *both* the temporal and spatial locality of missed cache blocks and determines whether to fetch *all* of the sectors within the cache block (*CG*-mode) or *only* the sectors that are requested (*FG*-mode). The key observation behind *BGP* is that for cache blocks with poor temporal locality, it is sufficient to fetch only the sectors that are actually requested (on-demand) because the other sectors will most likely not be referenced during their lifetime (e.g., cache blocks with *zero* reuse in Figure 5). Meanwhile, blocks with both high temporal and spatial locality make effective use of coarse-grained accesses, such that a simple bi-modal prediction is sufficient to greatly improve memory system performance and efficiency.

3.4.2 Microarchitecture

Our lightweight *BGP* microarchitecture determines whether each missed cache block has enough locality to warrant a coarse-grained fetch. We implement the storage of *BGP* using a *bloom filter* [37] to minimize the cost of tracking the multitude of cache blocks in the system. A bloom filter is a space-efficient probabilistic data structure that is used to *test* whether an element is a member of a *set*. It consists of a bitarray with m-bits and n-hash functions. An element is *inserted* (Figure 6(a)) into the set by calculating n different hashes of the element and setting the corresponding bits of the bitarray. Testing if an element belongs to the set (Figure 6(b)) is done by feeding it to the n hash functions and checking if all the corresponding bit positions are 1s. If any of the queried bits are 0, the element is definitely not a member of the set (true negative) while all 1s indicates either that the element actually was inserted to the set (true positive) or that there are many hash collisions with other elements of the set (false positive). The false positive rate of a bloom filter, accordingly, is determined by the number and type of hash functions chosen and the size of the bitarray. For the purpose of the *BGP*, a bloom filter is used to track the set of evicted blocks having low (or high) locality, using their respective block address as the inserted element.

Table 1: Configuration parameters of *BGP* microarchitecture.

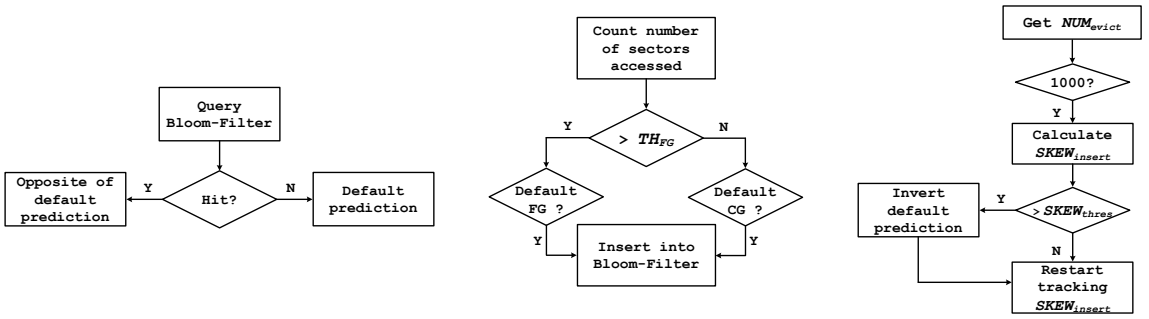
Bitarray size	2K-bit per bitarray (4K-bit per *BGP*)
Refresh period	Every 512 insertions
# of hash functions	6
Hash function	Byte-sliced XOR [39]
$\boldsymbol{TH_{FG}}$	2 sectors
$\boldsymbol{SKEW_{thres}}$	0.7

In order to temporally degrade old insertions and maintain a certain amount of locality history, the locality predictor is implemented using a *dual-bitarray* microarchitecture as detailed in Figure 6. This dual-bitarray bloom filter uses two temporally overlapped bitarrays that are periodically cleared and swapped in order to eliminate interference due to stale locality data. Such structure has several implementation advantages. First, the dual-bitarray bloom filter allows for the removal of aging elements in an application-appropriate manner without resorting to more expensive bloom filter variants (such as a counting filter [38]). Also, the rolling history of the dual-bitarray naturally captures temporal locality information. Finally, because the dual-bitarray structure periodically resets, it allows us to tailor the default insertion/prediction mode (*CG* or *FG*) to dynamic phase behavior in order to reduce the false positive rate, as described below.

3.4.3 Prediction Mechanism

Figure 7 summarizes how bi-modal granularity prediction operates and when and how evicted blocks are inserted into the bloom filter. The *BGP* contains a *default* prediction (*CG* or *FG*) that determines what kind of evicted blocks are inserted into the filter (and the corresponding prediction upon a query to the filter). The *CG*/*FG* fetch decision is made by querying the bloom filter with the evicted block's address—upon a miss, the querying cache block has the opposite locality characteristics to the blocks inserted into the bloom filter, so *BGP* grants the default prediction. For those queries that hit in the bloom filter, the *BGP* predicts the opposite of the default prediction (Figure 7(a)).

The *BGP* uses the number of sectors accessed as means to approximate the locality of cache block (rather than the number of accesses to the cache blocks) for simplicity in



(a) The CG/FG prediction mechanism. (b) The *BGP* insertion mechanism. (c) Default prediction inversion.

Figure 7: The prediction algorithm of the *BGP*. $SKEW_{insert}$ is evaluated every thousand cache block evictions (NUM_{evict}).

Table 2: Baseline GPGPU-Sim configuration.

Number of SMs	15
Threads per SM	1536
Threads per warp	32
SIMD lane width	32
Registers per SM	32768
Shared memory per SM	48KB
Warp scheduling policy	Oldest CTA First [7]
L1 cache (size/assoc/block size)	16KB/4-way/128B
L2 cache (size/assoc/block size)	768KB/16-way/128B
Number of memory channels	8
Memory bandwidth	179.2 GB/s
Memory controller	Out-of-order (FR-FCFS)

design. When a cache block is evicted, accordingly, the associated sector used-bit information is examined to estimate the block's locality—if the number of sectors accessed is below a pre-determined threshold (TH_{FG}), that block is estimated as having low locality and high locality otherwise (Figure 7(b)). This locality estimate is compared with the *BGP*'s current default prediction in order to determine whether the eviction should be inserted into the filter.

When the percentage of evicted cache blocks inserted into the bloom filter ($SKEW_{insert}$) is high, each filter will be filled quickly and the *BGP* will track little temporal history (Figure 6(c)). Inspired by the intuition of *agree predictors* [40], the *BGP* rotates the default prediction whenever $SKEW_{insert}$ is higher than a pre-determined threshold ($SKEW_{thres}$) in order to avoid the bloom filter from being rapidly saturated by an overwhelming number of insertions (Figure 7(c)). Table 1 summarizes the microarchitectural parameters used for the baseline *BGP* configuration. The *BGP* bloom filter hash functions are inexpensively implemented in hardware by byte-slicing each evicted address and *XOR*-ing the slices together [39]. Overall, we observe that the prediction accuracy of *BGP* is relatively insensitive to these parameters, unless the bitarray size is less than 2K-bits or the refresh period is less than a quarter of the bitarray size.

3.4.4 Summary of the Benefits of BGP

The benefits of the proposed *BGP* are twofold. First, by granting *FG* accesses for only those accesses that have past history of low temporal locality, applications with good caching behavior (e.g., most of the sectors are utilized) or those with a working set fitting well in the cache (e.g., low miss rates and thus low evictions within a timeframe) are guaranteed to fetch data in *CG*-mode, maintaining the benefits of the *CG*-only memory system. Second, the bloom filter based *BGP* provides a cost-effective mechanism to determine the access granularity, as opposed to SPP-based schemes that require a separate PHT and complex control/update logic.

4. METHODOLOGY

4.1 Simulation Model

We model *LAMAR* using GPGPU-Sim (version 3.2.0) [41, 42], which is a cycle-level performance simulator of a general purpose GPU architecture that supports CUDA 3.1 and its PTX ISA. The memory hierarchy of GPGPU-Sim is augmented with sectored L1/L2 caches and DrSim [43, 44], a detailed DRAM simulator that supports sub-ranked memory systems (Section 3). We configure our DRAM model to adhere to the GDDR5 specification [24], except for the bank-grouping effects (which are projected to be eliminated in future GDDR products [24, 45]). To demonstrate how the *BGP* is affected by limited hardware resources (e.g., dual 2K-bitarrays), we also simulate the SPP and *BGP* with unrealistically large histories (1M-entries); these impractical designs are denoted by SPP and BGP_{inf} henceforth.

In general, the GPU simulator is configured to be similar to NVIDIA's GTX480 [19] using the configuration file provided with GPGPU-Sim [46]. Key microarchitectural parameters of the baseline configuration are summarized in Table 2; we explicitly mention when deviating from these parameters for the sensitivity studies in Section 5.5.

4.2 GDDR5 Power Model

We model DRAM power based on the Hynix GDDR5 specification [24] as summarized in the equation below. Our power model includes the background power, refresh power (P_{REF}), activation & precharge power (P_{ACT_PRE}), read power (P_{RD}) and write power (P_{WR}). The background power includes precharge standby power (P_{PRE_STBY}) and active standby power (P_{ACT_STBY}). Read and write power includes the power consumed by the DRAM bank (P_{RD_BANK}) and by the IO pins (P_{RD_IO}).

$$P_{GDDR5} = \underbrace{P_{PRE\_STBY} + P_{ACT\_STBY}}_{Background\ Power} + P_{REF} + P_{ACT\_PRE} + \underbrace{P_{RD\_BANK} + P_{RD\_IO}}_{P_{RD}} + \underbrace{P_{WR\_BANK} + P_{WR\_IO}}_{P_{WR}}$$

4.3 GPU Processor Power Modeling

This study is concerned primarily with the performance and efficiency of the memory hierarchy. To evaluate how *LAMAR* affects the overall system energy-efficiency, however, we model the GPU processor power using the analytical IPC-based power model suggested by Ahn et al. [25]. The peak power consumption of each SM is extracted using GPUWattch [47]. The leakage power of the system (including GPU processors and DRAM) is estimated to be 59 Watts. The peak dynamic

Table 3: Evaluated CUDA benchmarks. Avg_{sec} refers to the average number of sectors accessed across all cache blocks.

Abbr.	Description	#Instr.	Avg_{sec}	Ref.
IIX	Inverted index	1.8B	1.09	[2]
SSSP	Shortest paths	1.5B	1.24	[3]
BFS1	Breadth first search	3B	1.25	[3]
SP	Survey propagation	1.3B	1.28	[3]
SSC	Similarity score	4.9B	1.46	[2]
BFS2	Breadth first search	469M	1.48	[1]
MUM	MUMmerGPU	149M	1.49	[49]
NW	Needleman-Wunsch	220M	1.67	[1]
PVC	Page view count	5.4B	1.75	[2]
WP	Weather prediction	365M	2.00	[41]
MST	Min. spanning tree	5B	2.39	[3]
RAY	Ray-tracing	750M	3.29	[41]
SCLST	Streamcluster	4.1B	3.41	[1]
BACKP	Back propagation	196M	3.62	[1]
NN	Neural network	78M	3.65	[41]
SRAD	Structured grid	8.5B	3.88	[1]
LAVAMD	N-body	22B	3.89	[1]
SPROD	Scalar-product	25M	3.99	[48]
MCARLO	Monte-carlo	1B	3.99	[48]
FWT	Fast-walsh-transform	3.9B	4.00	[48]

power consumption per SM is estimated to be 9.5 Watts, out of which 2.3 Watts belongs to constant power that does not scale with IPC. Such simple IPC-based power modeling offers $> 90\%$ agreement with GPUWattch and is used to estimate the overall system efficiency in Section 5.4.

4.4 Benchmarks

LAMAR is evaluated with 32 benchmarks from Rodinia [1], CUDA-SDK [48], MapReduce [2], LonestarGPU [3], and the benchmarks provided with GPGPU-Sim [41]. We report the 20 applications (Table 3) that exhibit noticeable differences across different schemes for brevity. All benchmarks are simulated to completion, with the exception of SSSP, SP, PVC, SCLST, and FWT—due to the long simulation time of these applications, we execute them only up to the point where IPC is saturated with small variation among different iterations of the kernel. We categorize the 20 chosen benchmarks as either being *FG-leaning* or *CG-leaning* based on the average number of sectors accessed within all L1/L2 cache blocks (Figure 1) — applications that average more than two sectors accessed per cache block are categorized as *CG*-leaning (and *FG*-leaning otherwise).

5. EVALUATION

This section evaluates *LAMAR*, considering its impact on cache efficiency, the improvements that *LAMAR* brings about in overall performance and energy-efficiency. We also discuss its sensitivity to key microarchitectural parameters and its implementation overheads. We compare five different GPU memory hierarchy designs: *CG*-only, *FG*-only, SPP, BGP_{inf} and *BGP*, which are denoted by C/F/S/I/B, respectively, in all figures throughout this section. A *LAMAR* configuration based on static GDU decisions is equivalent to the best of *CG*-only and *FG*-only for each application. Note that we used the same dataset for both the profiling and measurement run. All average values are based on harmonic means.

5.1 Prediction Quality, Traffic, and Caching Efficiency

The GDU of *LAMAR* determines whether a cache block should be fetched in *CG* or in *FG* mode. It can therefore predict to: 1) correctly fetch sectors that are actually referenced (*PRED_REF*), 2) incorrectly fetch sectors that are not referenced (*PRED_NREF*), and 3) incorrectly not fetch sectors that are referenced later (*NPRED_REF*). We therefore categorize each fetched sector as either being fetched on-demand from the upper level (*DEMAND*) or based on prediction (Figure 8)[5] The overall read/write traffic and the associated cache miss rates are depicted in Figure 9 and Figure 10. Overall, the *FG*-only scheme has the smallest off-chip traffic thanks to its conservative fetch decision. This reduced traffic, however, comes at the cost of a significant portion of sectors being *NPRED_REF* with increased miss rates for some applications. NN, for instance, contains 65%/54% of its L1/L2 sectors being fetched *NPRED_REF*. Because these sectors would have been pre-fetched had the initial access been predicted as *CG*-fetches, memory access behavior and caching efficiency are degraded, potentially leading to performance penalties for certain applications (see Section 5.2 for details). *CG*-leaning applications generally contain less overfetched data (even with the *CG*-only scheme), with only 13%/3% more sectors fetched to L1/L2 compared to the *FG*-only scheme. For *FG*-leaning benchmarks, however, *CG*-only falls short by having 171% and 93% more L1/L2 read-in traffic than *FG*-only, most of which is due to the large number of mispredicted *PRED_NREF* sectors.

Dynamically-driven *LAMAR*, on the other hand, is able to balance the benefits of both *CG*-only and *FG*-only schemes. All three *LAMAR* predictors can reduce off-chip traffic significantly without degrading the memory access behavior of *CG*-leaning applications. SPP is the least effective mechanism among the three, having 60%/72% more L1/L2 read-in sectors than *FG*-only, whereas BGP_{inf} and *BGP* contain 20%/22% and 37%/47% more, respectively, thanks to the GPU-context appropriate prediction algorithm (Section 3.4).

In general, the *CG*-only scheme falls short by significantly overfetching data for *FG*-leaning applications while the *FG*-only scheme (despite its advantage in reducing off-chip traffic) disrupts the memory access behavior of several benchmarks. Accordingly, we observe that a static GDU configuartion, preferrably matching application characteristics (e.g., Avg_{sec} provided by the profiler/autotuner), typically performs best in terms of overall bandwidth utilization, maximizing energy-efficiency. While less effective than the best-performing *CG*-/*FG*-only scheme for each application, dynamically-driven *LAMAR* approximates the characteristics of the static GDU schemes, balancing the benefits of *CG*-fetches while reducing traffic when feasible. Compared to the *BGP*, SPP-based prediction lacks robustness and fails to effectively reduce off-chip traffic for SSSP, BFS1, SP, MUM and NW. However, SPP is still advantageous compared to a static memory hierarchy.

5.2 Performance

Figure 11 shows the overall speedup from adopting *LAMAR* memory schemes. In general, all *LAMAR* predictors provide significant benefit over the conventional *CG*-only memory system while executing *FG*-leaning applications thanks to more efficient utilization of the off-chip bandwidth, demonstrating a maximum 49% boost and an average 12–14% improvement in performance. *LAMAR* predictors also provides comparable performance to the *CG*-only scheme in executing *CG*-leaning applications—the biggest degradation is for MST (whose L1

[5]Note that sectors requested from the L1 to L2 cache are interpreted as *DEMAND* sectors from L2's perspective, even though these sectors can be *PRED_REF*, *PRED_NREF*, and *NPRED_REF* from the L1's point of view.

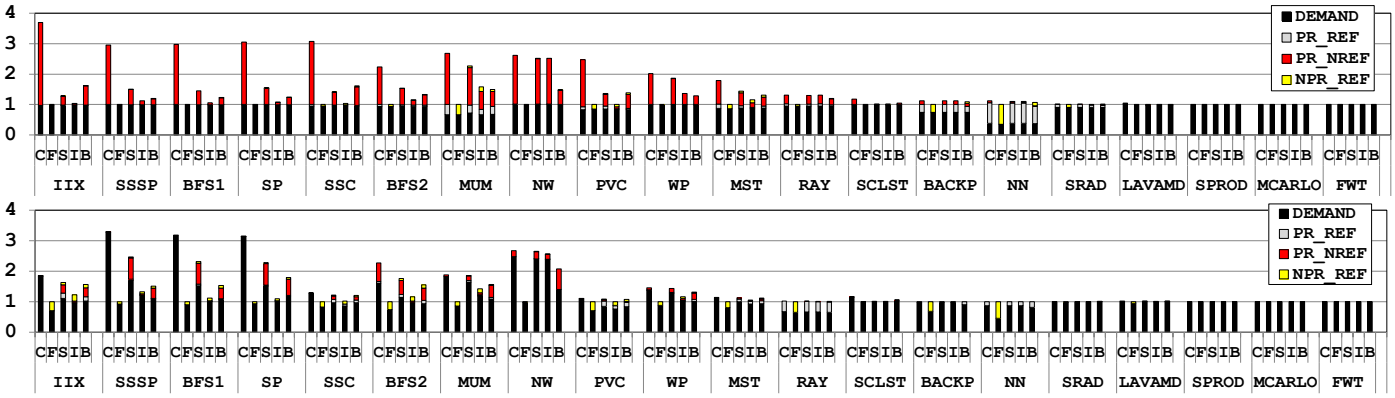


Figure 8: The number of sectors read into the L1 (top) and L2 (bottom) caches, categorized based on prediction quality (normalized to the FG-only scheme).

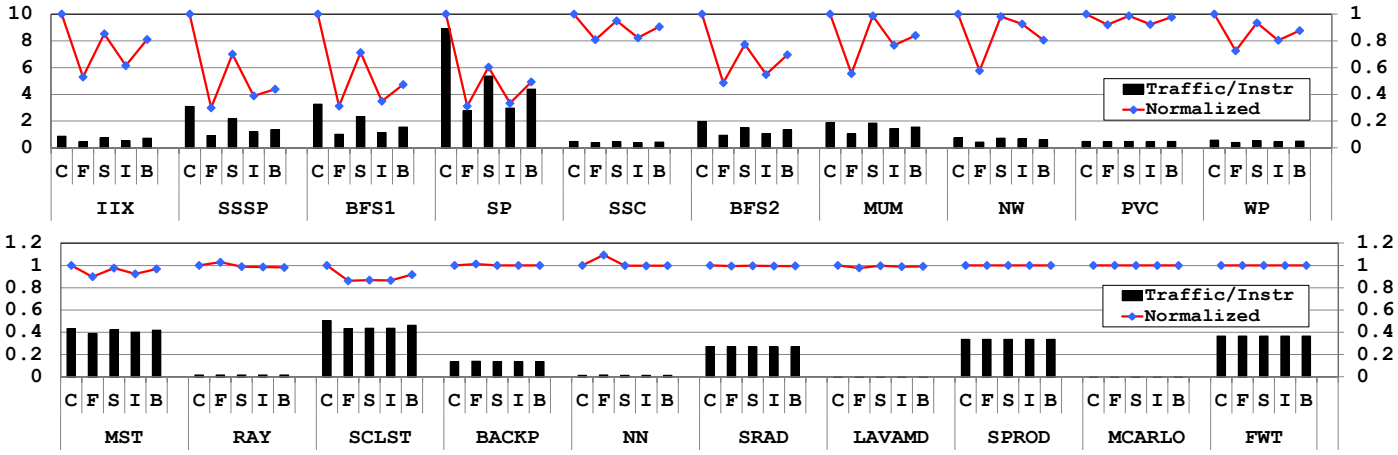


Figure 9: Byte traffic to DRAM (both read/write) normalized by 1) the number of instructions (left axis) and 2) by the traffic/instr. of the CG-only scheme (right axis).

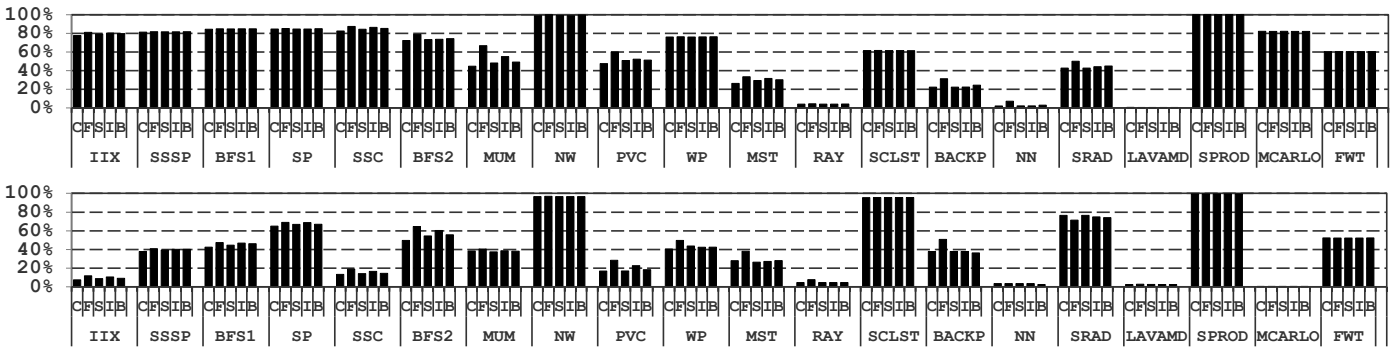


Figure 10: The cache miss rates of the L1 (top) and L2 (bottom) caches.

caching efficiency is disrupted by *LAMAR*, lowering the IPC by 13% with BGP_{inf}). The static *FG*-only scheme adversely impacts 5 of the *CG*-leaning benchmarks, ranging from 4% (SPROD) to 22% (MST) degraded performance.

5.3 Impact on DRAM Power Efficiency

Correctly predicted *FG*-fetches reduce the number of read and write commands issued to DRAM. However, *CG*-fetches have the advantage of leveraging DRAM bank row locality by only having to open the corresponding bank row once. This is not the case for mispredicted *FG*-fetches (*NPRED_REF*), which require re-opening the bank row at a later time and lead to additional activate/precharge (ACT/PRE) commands. Figure 12 illustrates how the reduction in off-chip traffic correlates with DRAM power consumption. Overall, the benefit of reduced read/write commands outweighs the overhead of increased ACT/PRE commands. For *FG*-leaning applications, *FG*-only achieves the largest average power reduction of 19% (max 42%) while SPP/BGP_{inf}/*BGP* obtain an average 1%/13%/8% reduction (max 16%/39%/33%), respectively.

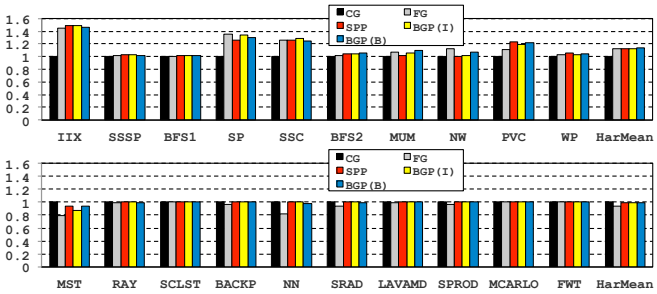


Figure 11: Normalized speedup. BGP(I) and BGP(B) represent BGP_{inf} and BGP.

Despite its low implementation cost, BGP is competitive with BGP_{inf} and outweighs SPP both in power reduction as well as Performance/Watt (*Perf/Watt*), with an average 27% increase in *Perf/Watt* while SPP and BGP_{inf} achieve an average 16%/34% improvement, respectively.

For CG-leaning applications, all $LAMAR$ predictors perform comparable to the CG-only scheme whereas FG-only suffers from an average 5% degradation in *Perf/Watt* (maximum 20% degradation).

5.4 System-Level Power Efficiency

We evaluate the system-level efficiency of different memory schemes by combining the DRAM power model (Section 5.3) with the IPC-based GPU processor power model (Section 4). Recent literature [47, 50] estimates that the memory system consumes approximately 5 to 45% of the overall GPU power, depending on the application. $LAMAR$ mainly improves the energy-efficiency of the memory hierarchy, so the overall improvement in *Perf/Watt* is less pronounced than its DRAM counterpart. Among FG-leaning applications, BGP_{inf} and BGP obtains an average 18%/17% improvement in *Perf/Watt* respectively. SPP helps the least among $LAMAR$ predictors with an average 13% improvement in *Perf/Watt*. The FG-only mechanism, while achieving the highest average *Perf/Watt* improvement (19%), struggles in executing CG-leaning applications and significantly degrades *Perf/Watt* for MST, NN, and SRAD.

5.5 Sensitivity Study

We summarize $LAMAR$'s sensitivity to key parameters in this subsection. Due to space limitations, we mainly discuss the effectiveness of $LAMAR$ on FG-leaning applications.

5.5.1 Cache Capacity

As shown in Figure 14(a), DRAM traffic is generally reduced with larger on-chip caches (and vice versa for smaller caches) thanks to better caching efficiency. The benefits of $LAMAR$ are still maintained across all FG-leaning applications and the relative reduction in traffic (compared to each configuration's CG-only scheme) is more pronounced with smaller caches (e.g., IIX, BFS1, SSC, BFS2, WP). BGP, for instance, provides an average 37%/33%/17% reduction in traffic with the three cache size configurations.

5.5.2 Cache Block Size

With a larger L2 cache block (256B), the baseline CG-only scheme is likely to overfetch even more off-chip sectors and to suffer from severe bandwidth under-utilization. Such behavior is illustrated in Figure 14(b) where a 256B configuration of CG-only uses an average 96% more memory traffic than the baseline. The benefits of $LAMAR$, accordingly, are much more evident under the 256B configuration, where BGP reduces off-chip traffic by an average of 58% compared to the 33% reduction using the baseline cache block size.

5.5.3 Larger Sector Size

To demonstrate the benefits of $LAMAR$ with minimal changes to the GPU system, we evaluate the proposed mechanisms without a sub-ranked off-chip memory system. A conventional GDDR5-based memory system provides a minimum access granularity of 64-bytes (Figure 2), so we evaluate $LAMAR$ with a 64-byte sector size and minimum access granularity. As depicted in Figure 14(c), the benefit of $LAMAR$ is reduced from an average 33% traffic reduction to 20% under BGP due to the lack of sub-ranking.

5.5.4 Thread-Level Parallelism

Recent literature [36, 10] shows that throughput processors make poor use of data caches, due to the high cache access intensity and the resulting low per-thread cache capacity. To this end, previous work makes the warp scheduler *cache-conscious* [10] such that the number of warps able to access the cache are dynamically reduced (hence throttling thread-level parallelism [TLP] available at the SM) if the cache is thrashing. Such cache-conscious warp scheduling (CCWS) is therefore only effective when the application is both cache-sensitive and is thrashing. While $LAMAR$ focuses on wasted-transfers due to granularity mismatches in the system and is orthogonal to CCWS, we nonetheless evaluate the effectiveness of $LAMAR$ on top of this technique. Since CCWS is effectively a dynamic mechanism that approximates the statically chosen optimal level of TLP, we experiment $LAMAR$ on top of CCWS as detailed in Figure 15. As depicted, three of the 20 applications we study (IIX, MUM, SCLST) benefit from TLP throttling and $LAMAR$ remains effective in the presence of TLP tuning.

5.5.5 Miscellaneous

As mentioned in Section 3.4, the prediction quality of the baseline BGP microarchitecture is relatively robust with bitarray sizes larger than 2K-bits. Performance is improved by 2% to 7% with a 4K-bitarray, but saturates when going from 4K-bits to 8K-bits. Changing TH_{FG} and $SKEW_{thres}$ (Figure 7) also affects off-chip traffic and performance, but overall trends remain similar to the analysis discussed throughout this section (so long as $SKEW_{thres}$ is above 0.7 and TH_{FG} is less than three sectors).

5.6 Implementation Overhead

$LAMAR$ is implemented using a sector cache and a sub-ranked memory system, the overheads of which are well established in previous literature [25, 26, 27, 28, 29, 30]. In addition, each cache partition is augmented with a GDU. Static GDU configurations require no additional hardware, but necessitate profiler/autotuner support to provide recommended granularity information. We leave further exploration of identifying the optimal granularity to future work.

For dynamic GDU schemes, the proposed BGP microarchitecture (using a dual-bitarray bloom filter) requires 1) $4K$-bits of storage per GDU, 2) 6 sets of XOR logic gates for the hash functions, and 3) control logic to insert/test the membership of the bloom filter (Table 1).

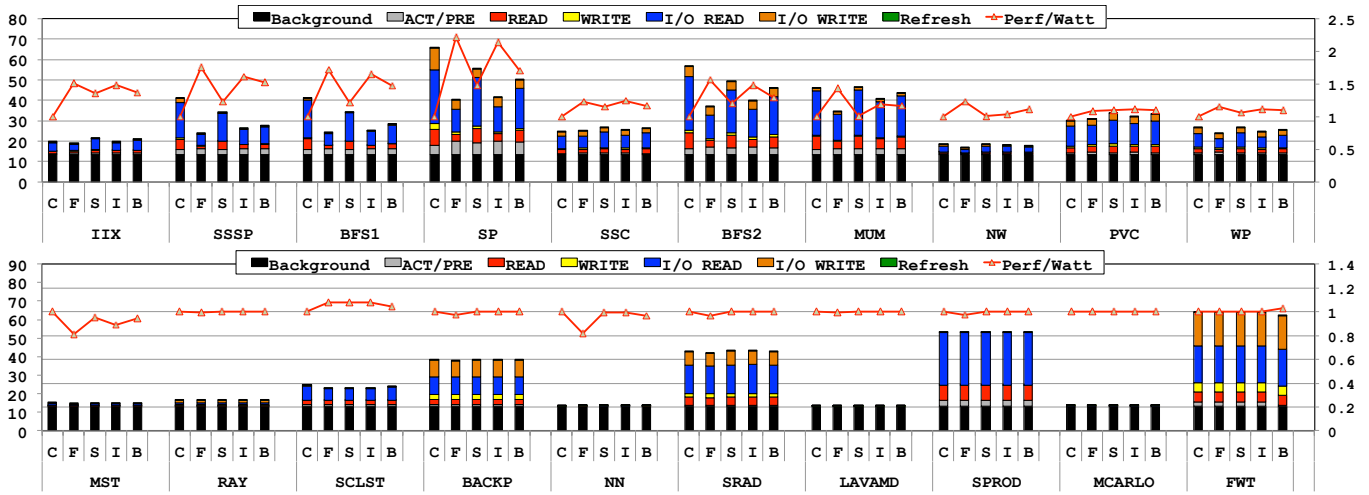


Figure 12: A breakdown of DRAM power (Watt, left axis) and the corresponding *Perf/Watt* (normalized to *CG*-only, right axis).

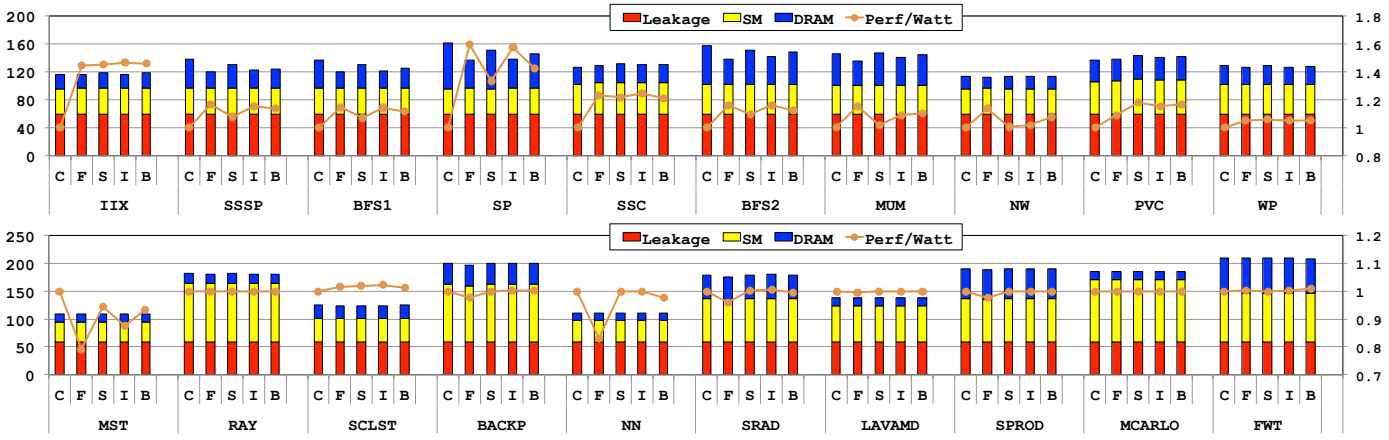


Figure 13: The system power consumption (Watt, left axis) and the corresponding *Perf/Watt* (normalized to *CG*-only, right axis). Note that the range of the upper plot is 0.8–1.8, whereas it is only 0.7–1.2 for the bottom plot.

6. DISCUSSION

6.1 LAMAR on Future Memory Technologies

In order to be concrete and to allow a detailed evaluation, we use GDDR5 as our memory technology. Future GPUs, however, may use evolving memory interface standards that utilize 3D packaging technology, such as the *Hybrid Memory Cube* (HMC) [51] or *High-Bandwidth Memory* (HBM) [52]. Although these interfaces are likely to offer much higher bandwidth than GDDR5, GPU arithmetic performance will increase as well, such that effectively utilizing memory throughput will remain critical to performance and efficiency. In fact, capacity-to-bandwidth ratio is likely to increase with the use of HMC packages—this implies that bandwidth utilization will increase in importance, amplifying the potential benefits of *LAMAR*. The proposed access granularity for these interfaces is also similar to that of GDDR5 devices today, with HMC proposing an access granularity of $32 - 256$-bytes and with HBM likely to use a 32-byte granularity similar to the WideIO standard [53]. Thus, the opportunity and policies we propose for *LAMAR* should generally apply equally well.

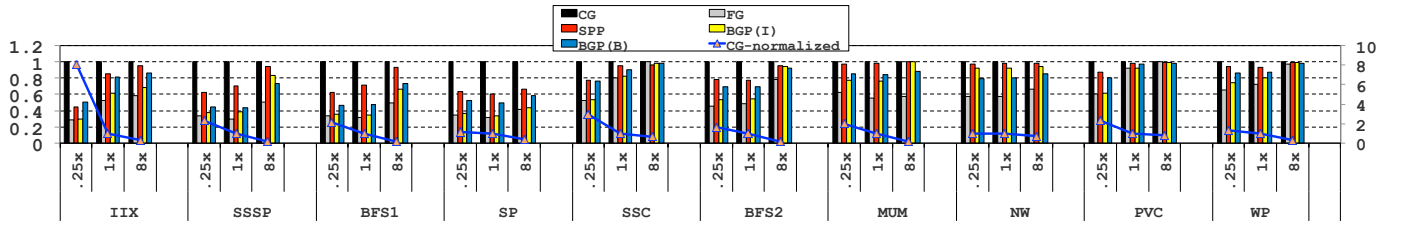
With 3D packaging, it is likely that the memory controller will be partitioned between the processor and the DRAM die stacks: scheduling is likely to remain close to the processor, where knowledge of priorities and requests is readily available, while implementation of the DRAM access protocol will be relegated to the controller within each stack [51]. This partitioning will require a re-design of how *LAMAR* controls the memory modules (owing to the fact that sub-ranks are essentially internalized and hidden within each stack). Because scheduling is still delegated to the processor, *LAMAR* will have to be modified to account for sending the appropriate request packets to maximize transfer efficiency. We will develop and analyze such designs in future work.

6.2 Error Correction

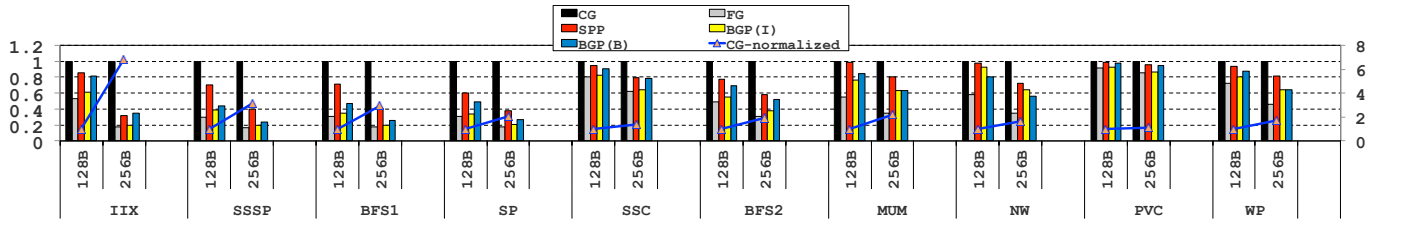
Current GPUs support error correction with error correcting codes (ECC). While the details of the memory protection schemes in industry are not publicly known, one way of flexibly supporting error correction without dedicated DRAM chips is through virtualized ECC [54]. The approach taken by *LAMAR* is amenable to such error correction, in a similar manner to prior work [18, 32].

6.3 Alternative FG Cache Management

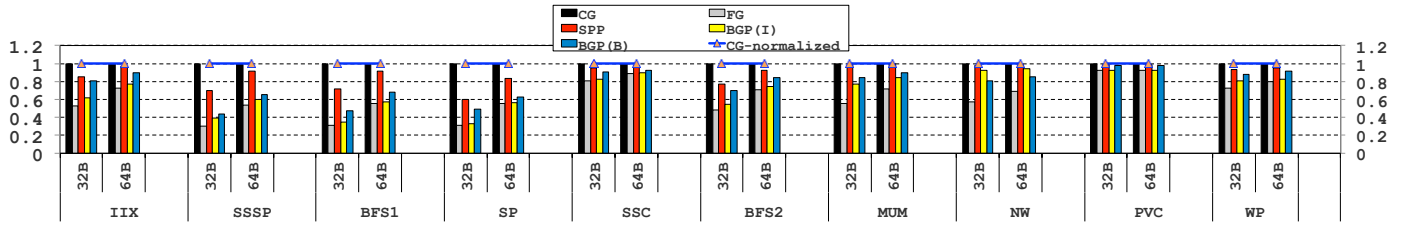
LAMAR uses a simple sector cache to manage *FG* data in the on-chip cache hierarchy, as our current study focuses on



(a) The sensitivity of *LAMAR* to reduced (0.25 times) and increased (8 times) L1/L2 cache capacity.



(b) The sensitivity of *LAMAR* to L2 cache block size. The overall L2 cache capacity is maintained equal to the baseline.



(c) The sensitivity of *LAMAR* to the minimum access granularity (32-byte/64-byte). Note that the L1/L2 cache capacity and cache block size are maintained equal to the baseline.

Figure 14: The sensitivity of off-chip traffic to differing (a) L1/L2 cache capacities, (b) L2 cache block sizes, and (c) minimum access granularities. The left axis represents the off-chip traffic of five different memory hierarchies, normalized to each configuration's *CG*-only scheme. The right axis is used to compare the traffic of each configuration's *CG*-only scheme (*CG-normalized*) and is normalized to the baseline configuration (Table 2).

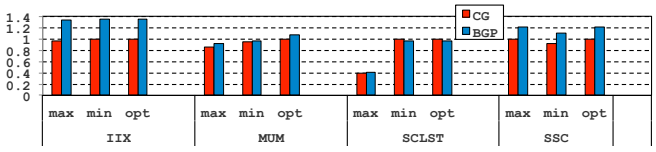


Figure 15: Sensitivity of *LAMAR* to available TLP. We experiment by sweeping through the number of schedulable CTAs within an SM from one (*min*) to its maximum allowable number (*max*), which is limited by available hardware resources. The speedup with the optimal number of CTAs is reported as *opt*. The figure shows the IPC normalized to the *CG*-only scheme with *opt* TLP. Only three applications (IIX, MUM, SCLST) benefit from TLP throttling, meaning that, the best performance is generally achieved with the maximum level of TLP. Such common-case application behavior is represented by SSC whose performance is degraded with reduced TLP.

the efficient management of off-chip data for irregular GPU applications. However, alternative *FG* cache management schemes exist, such as the decoupled sectored cache [55], pool-of-sectors cache [56], or the spatio/temporal cache [57]. Such more advanced cache architectures could be adopted to increase the *effective capacity* of the cache for irregular applications. Some irregular applications are very sensitive to the (typically limited) on-chip storage capacity, such that these alternative caches could significantly increase performance. Optimization of the on-chip portion of *LAMAR* is left for future study.

6.4 Other Dynamic Bloom-Filter Mechanisms

The *BGP* incorporates two temporally-separated Bloom filters to support the aging of membership data and to allow space-efficient operation with a dynamic stream of accesses. The temporal aging of Bloom filter entries for dynamic data has been addressed by Deng and Rafiei [58] by associating a slowly degrading count with each storage cell. However, this design makes inefficient use of storage and is unlikely to perform competitively with the *BGP*. The concept of maintaining and swapping two temporally-separated Bloom filters has been previously employed in software for filtering dynamic data [59, 60, 61]. *BGP* is the first application of such a scheme to memory access granularity prediction in hardware and is unique in its implementation and default prediction inverting algorithm. Yoon [62] recently proposed an alternate two-buffer algorithm for filtering dynamic data that could provide modest accuracy benefits for the *BGP*. Incorporating and analyzing this algorithm (along with *BGP*'s novel default prediction inversion algorithm) is left for future work.

7. CONCLUSIONS

The increasing popularity of general-purpose GPU programming and the growing irregularity of throughput-oriented programs necessitate a fine-grained GPU memory system. Meanwhile, the continuing need for the high-performance acceleration of regular, well structured programs and graphical workloads make coarse-grained memory accesses compulsory as well. This paper proposes *LAMAR*, an adaptive and reactive hardware-only memory scheme for GPUs and

throughput-oriented processors that achieves superior efficiency across a range of general-purpose GPU applications. By dynamically predicting the temporal and spatial locality of memory accesses, *LAMAR* mitigates the deficiencies of static-granularity memory systems and prior mixed-granularity memory schemes for control-intensive CPUs. In addition, the hardware required for *LAMAR* is simple and non-intrusive enough to be readily implemented in a many-core GPU and its adoption requires no programmer intervention. Our results show that *LAMAR* provides an average 14% increase in performance (max 49%), 33% reduction in average off-chip traffic (max 64%), and an average 17% improvement in system-level energy-efficiency (max 47%).

8. ACKNOWLEDGMENTS

We would like to thank the developers of GPGPU-Sim and the anonymous reviewers, who provided excellent feedback for preparing the final version of this paper. Special thanks to Min Kyu Jeong at Oracle Labs for answering numerous questions regarding the use of DrSim. This work is supported in part by the National Science Foundation (Grant #0954107) and Intel Labs URO for the Memory Hierarchy Innovations Program.

9. REFERENCES

[1] S. Che, M. Boyer, J. Meng, D. Tarjan, J. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *IEEE International Symposium on Workload Characterization (IISWC-2009)*, October 2009.

[2] B. He, W. Fang, Q. Luo, N. Govindaraju, and T. Wang, "A MapReduce Framework on Graphics Processors," in *17th International Conference on Parallel Architecture and Compilation Techniques (PACT-17)*, 2008.

[3] M. Burtscher, R. Nasre, and K. Pingali, "A Quantitative Study of Irregular Programs on GPUs," in *IEEE International Symposium on Workload Characterization (IISWC-2012)*, 2012.

[4] W. W. Fung, I. Sham, G. Yuan, and T. M. Aamodt, "Dynamic Warp Formation and Scheduling for Efficient GPU Control Flow," in *40th International Symposium on Microarchitecture (MICRO-40)*, December 2007.

[5] D. Tarjan, J. Meng, and K. Skadron, "Increasing memory miss tolerance for SIMD cores," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis (SC-09)*, 2009.

[6] J. Meng, D. Tarjan, and K. Skadron, "Dynamic warp subdivision for integrated branch and memory divergence tolerance," in *37th International Symposium on Computer Architecture (ISCA-37)*, 2010.

[7] W. W. Fung and T. M. Aamodt, "Thread Block Compaction for Efficient SIMT Control Flow," in *17th International Symposium on High Performance Computer Architecture (HPCA-17)*, February 2011.

[8] V. Narasiman and et al., "Improving GPU Performance via Large Warps and Two-Level Warp Scheduling," in *44th International Symposium on Microarchitecture (MICRO-44)*, December 2011.

[9] M. Rhu and M. Erez, "CAPRI: Prediction of Compaction-Adequacy for Handling Control-Divergence in GPGPU Architectures," in *39th International Symposium on Computer Architecture (ISCA-39)*, June 2012.

[10] T. Rogers, M. O'Connor, and T. Aamodt, "Cache-Conscious Wavefront Scheduling," in *45th International Symposium on Microarchitecture (MICRO-45)*, December 2012.

[11] M. Rhu and M. Erez, "The Dual-Path Execution Model for Efficient GPU Control Flow," in *19th International Symposium on High-Performance Computer Architecture (HPCA-19)*, February 2013.

[12] A. Jog and et al., "OWL: Cooperative Thread Array Aware Scheduling Techniques for Improving GPGPU Performance," in *13th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-13)*, 2013.

[13] M. Rhu and M. Erez, "Maximizing SIMD Resource Utilization in GPGPUs with SIMD Lane Permutation," in *40th International Symposium on Computer Architecture (ISCA-40)*, June 2013.

[14] A. Vaidya and et al., "SIMD Divergence Optimization through Intra-Warp Compaction," in *40th International Symposium on Computer Architecture (ISCA-40)*, June 2013.

[15] A. Jog and et al., "Orchestrated Scheduling and Prefetching for GPGPUs," in *40th International Symposium on Computer Architecture (ISCA-40)*, 2013.

[16] S. Keckler, W. Dally, B. Khailany, M. Garland, and D. Glasco, "GPUs and the Future of Parallel Computing," in *IEEE Micro*, October 2011.

[17] D. H. Yoon, M. K. Jeong, and M. Erez, "Adaptive granularity memory systems: A tradeoff between storage efficiency and throughput," in *38th International Symposium on Computer Architecture (ISCA-38)*, 2011.

[18] D. H. Yoon, M. Sullivan, M. K. Jeong, and M. Erez, "The dynamic granularity memory system," in *39th International Symposium on Computer Architecture (ISCA-39)*, 2012.

[19] NVIDIA Corporation, "NVIDIA's Next Generation CUDA Compute Architecture: Fermi," 2009.

[20] ——, "Whitepaper: NVIDIA GeForce GTX 680," 2012.

[21] AMD Corporation, "AMD Radeon HD 6900M Series Specifications," 2010.

[22] NVIDIA Corporation, "NVIDIA CUDA Programming Guide," 2011.

[23] AMD Corporation, "ATI Stream Computing OpenCL Programming Guide," August 2010.

[24] *1Gb (32Mx32) GDDR5 SGRAM, H5GQ1H24AFR*, Hynix, 2009.

[25] J. H. Ahn, N. P. Jouppi, C. Kozyrakis, J. Leverich, and R. S. Schreiber, "Future scaling of processor-memmory interfaces," in *Proc. the Int'l Conf. High Performance Computing, Networking, Storage and Analysis (SC)*, Nov. 2009.

[26] J. H. Ahn, J. Leverich, R. Schreiber, and N. P. Jouppi, "Multicore DIMM: An energy efficient memory module with independently controlled DRAMs," *IEEE Computer Architecture Letters*, vol. 8, no. 1, pp. 5–8, Jan. - Jun. 2009.

[27] F. A. Ware and C. Hampel, "Micro-threaded row and column operations in a DRAM core," in *Proc. the first*

Workshop on Unique Chips and Systems (UCAS), Mar. 2005.

[28] ——, "Improving power and data efficiency with threaded memory modules," in *Proceedings of the International Conference on Computer Design (ICCD)*, 2006.

[29] H. Zheng and et al., "Mini-rank: Adaptive DRAM architecture for improving memory power efficiency," in *41st International Symposium on Microarchitecture (MICRO-41)*, Nov. 2008.

[30] T. M. Brewer, "Instruction set innovations for the Convey HC-1 computer," *IEEE Micro*, vol. 30, no. 2, pp. 70–79, 2010.

[31] J. S. Liptay, "Structural aspects of the system/360 model 85, part II: The cache," *IBM Systems Journal*, vol. 7, pp. 15–21, 1968.

[32] S. Li and et al., "Mage: adaptive granularity and ecc for resilient and power efficient memory systems," in *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*. IEEE, 2012, pp. 1–11.

[33] D. Abts and et al., "The Cray Black Widow: A highly scalable vector multiprocessor," in *Proc. the Int'l Conf. High Performance Computing, Networking, Storage, and Analysis (SC)*, Nov. 2007.

[34] S. Kumar and C. Wilkerson, "Exploiting spatial locality in data caches using spatial footprints," in *25th International Symposium on Computer Architecture (ISCA-25)*, 1998.

[35] C. Chen, S.-H. Yang, B. Falsafi, and A. Moshovos, "Accurate and complexity-effective spatial pattern prediction," in *10th International Symposium on High Performance Computer Architecture (HPCA-10)*, 2004.

[36] W. Jia, K. Shaw, and M. Martonosi, "Characterizing and Improving the Use of Demand-Fetched Caches in GPUs," in *26th International Supercomputing Conference (ICS'26)*, 2012.

[37] B. Bloom, "Space/Time Trade-Offs in Hash Coding with Allowable Errors," in *ACM Communications*, 1970.

[38] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary cache: a scalable wide-area web cache sharing protocol," *IEEE/ACM Transactions on Networking (TON)*, vol. 8, no. 3, pp. 281–293, 2000.

[39] M. Ramakrishna and et al., "Efficient Hardware Hashing Functions for High Performance Computers," in *IEEE Transactions on Computers*, 1997.

[40] E. Sprangle, R. S. Chappell, M. Alsup, and Y. N. Patt, "The agree predictor: A mechanism for reducing negative branch history interference," in *17th International Symposium on Computer Architecture (ISCA-17)*, 1997.

[41] A. Bakhoda, G. Yuan, W. Fung, H. Wong, and T. Aamodt, "Analyzing CUDA workloads using a detailed GPU simulator," in *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS-2009)*, April 2009.

[42] "GPGPU-Sim," http://www.gpgpu-sim.org.

[43] "DrSim," http://lph.ece.utexas.edu/public/DrSim.

[44] M. K. Jeong, D. H. Yoon, D. Sunwoo, M. Sullivan, I. Lee, and M. Erez, "Balancing DRAM Locality and Parallelism in Shared Memory CMP Systems," in *18th International Symposium on High Performance Computer Architecture (HPCA-18)*, February 2012.

[45] T.-Y. Oh and et al., "A 7Gb/s/pin 1 Gbit GDDR5 SDRAM With 2.5 ns Bank to Bank Active Time and No Bank Group Restriction," in *IEEE Journal of Solid-State Circuits*, 2011.

[46] "GPGPU-Sim Manual," http://www.gpgpu-sim.org/manual.

[47] J. Leng and et al., "GPUWattch: Enabling Energy Optimizations in GPGPUs," in *40th International Symposium on Computer Architecture (ISCA-40)*, June 2013.

[48] NVIDIA Corporation, "CUDA C/C++ SDK CODE Samples," 2011.

[49] M. Schatz, C. Trapnell, A. Delcher, and A. Varshney, "High-throughput sequence alignment using graphics processing units," *BMC Bioinformatics*, vol. 8, no. 1, p. 474, 2007.

[50] M. Gebhart, D. Johnson, D. Tarjan, S. Keckler, W. Dally, E. Lindholm, and K. Skadron, "Energy-efficient mechanisms for managing thread context in throughput processors," in *38th International Symposium on Computer Architecture (ISCA-38)*, 2011.

[51] HMC, "Hybrid memory cube specification 1.0," Hybrid Memory Cube Consortium, 2013.

[52] Hynix, "Blazing a trail to high performance graphics," Hynix Semiconductor, Inc., 2011.

[53] JEDEC, "JESD 229 Wide I/O SDR," 2011.

[54] D. H. Yoon and M. Erez, "Virtualized and flexible ECC for main memory," in *Proc. the 15th Int'l. Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Mar. 2010.

[55] A. Seznec, "Decoupled sectored caches: Conciliating low tag implementation cost," in *Proc. the 21st Ann. Int'l Symp. Computer Architecture (ISCA)*, Apr. 1994.

[56] J. B. Rothman and A. J. Smith, "The pool of subsectors cache design," in *Proc. the 13th Int'l Conf. Supercomputing (ICS)*, Jun. 1999.

[57] A. Gonzalez, C. Aliagas, and M. Valero, "A data cache with multiple caching strategies tuned to different types of locality," in *Proc. the Int'l Conf. Supercomputing (ICS)*, Jul. 1995.

[58] F. Deng and D. Rafiei, "Approximately detecting duplicates for streaming data using stable bloom filters," in *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*. ACM, 2006, pp. 25–36.

[59] "bcache: A Linux kernel block layer cache," http://bcache.evilpiepirate.org/.

[60] F. Chang, W.-c. Feng, and K. Li, "Approximate caches for packet classification," in *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, vol. 4. IEEE, 2004, pp. 2196–2207.

[61] C. Ungureanu and et al., "TBF: A memory-efficient replacement policy for flash-based caches," in *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*, 2013, pp. 1117–1128.

[62] M. Yoon, "Aging bloom filter with two active buffers for dynamic sets," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 22, no. 1, pp. 134–138, 2010.