International Journal of Parallel Programming
https://doi.org/10.1007/s10766-022-00729-2

A Quantitative Study of Locality in GPU Caches for Memory-Divergent Workloads

Sohan Lal[1,2] · **Bogaraju Sharatchandra Varma**[3] · **Ben Juurlink**[2]

Received: 15 April 2021 / Accepted: 9 March 2022

Abstract
GPUs are capable of delivering peak performance in TFLOPs, however, peak performance is often difficult to achieve due to several performance bottlenecks. Memory divergence is one such performance bottleneck that makes it harder to exploit locality, cause cache thrashing, and high miss rate, therefore, impeding GPU performance. As data locality is crucial for performance, there have been several efforts to exploit data locality in GPUs. However, there is a lack of quantitative analysis of data locality, which could pave the way for optimizations. In this paper, we quantitatively study the data locality and its limits in GPUs at different granularities. We show that, in contrast to previous studies, there is a significantly higher inter-warp locality at the L1 data cache for memory-divergent workloads. We further show that about 50% of the cache capacity and other scarce resources such as NoC bandwidth are wasted due to data over-fetch caused by memory divergence. While the low spatial utilization of cache lines justifies the sectored-cache design to only fetch those sectors of a cache line that are needed during a request, our limit study reveals the lost spatial locality for which additional memory requests are needed to fetch the other sectors of the same cache line. The lost spatial locality presents opportunities for further optimizing the cache design.

Keywords Data locality · GPU caches · Memory divergence

✉ Sohan Lal
sohan.lal@tuhh.de

Bogaraju Sharatchandra Varma
s.bogaraju@ulster.ac.uk

Ben Juurlink
b.juurlink@tu-berlin.de

[1] Technische Universität Hamburg, Hamburg, Germany

[2] Technische Universität Berlin, Berlin, Germany

[3] Ulster University, Jordanstown, UK

Published online: 05 April 2022

1 Introduction

GPUs have been very successful in accelerating general-purpose applications from different domains. The advent of hardware-managed caches further accelerated the use of GPUs for general-purpose computing. Caches reduce off-chip memory traffic and cut down pressure on memory bandwidth by exploiting temporal and spatial locality, however, using caches efficiently for GPUs is a difficult task because a GPU employs a large number of threads, and GPU caches are small that lead to high contention and thrashing. In particular, there is a class of applications known as irregular applications that poorly utilize GPU caches. Irregular applications have memory divergence and/or control divergence, and they span a broad range of domains. GPUs issue concurrent memory accesses to consecutive addresses by coalescing memory accesses of threads in a warp. However, it may not always be possible to coalesce individual thread accesses due to scattered accesses that lead to memory divergence.

Memory divergence is known to cause many problems, including data over-fetch which can waste cache capacity, consume scarce resources such as miss status holding registers (MSHRs) and memory bandwidth, thus, further making it hard to utilize caches efficiently. As exploiting data locality is important for performance, there have been several studies to exploit the data locality in GPUs [9, 21]. Moreover, the recent generations of GPUs have adapted the cache design to tackle issues such as data over-fetch. For example, Maxwell, Pascal, and Volta GPU architectures use sectored caches [9] to fetch only the sectors that are requested instead of always fetching all sectors of a cache line. While sectored caches reduce data over-fetch, significant opportunities to exploit locality may also be lost as full locality information is not known at the time of a request. Moreover, there is a lack of quantitative analysis of data locality and data reuse in GPUs that can in general be useful for further optimizing a GPU architecture. Therefore, in this paper, we quantitatively study the data locality and its limits in GPUs at different granularities. We show that there is a higher locality in GPU caches than currently exploited by sectored caches, offering opportunities for further optimizing the cache design. Our study focuses on NVIDIA architectures, which dominate the discrete GPU market, however, we believe the findings mostly apply to AMD architectures as well because both of them have a similar cache hierarchy.

In summary, we make the following main contributions:

- This is the first comprehensive, quantitative, and limit study of locality in GPU data caches for memory-divergent workloads.
- We thoroughly study temporal and spatial locality at warp and thread-block granularities, quantifying contributions to overall locality.
- We show significantly higher inter-warp hits (46%) at the L1 cache for memory-divergent workloads compared to the state-of-the-art [21].
- We show that for memory-divergent workloads about 50% of the cache capacity, and other scarce resources such as NoC bandwidth, memory bandwidth are wasted due to data over-fetch.

- Our analysis shows that 57% of the cache lines are never re-referenced. However, the limit study shows that actually, only 30% of the cache lines are never re-referenced and 27% are evicted before re-reference.
- We show that the low spatial utilization justifies the sectored-cache design, however, the limit study reveals the lost spatial locality for which additional memory requests are needed in a sector cache and hence, showing opportunities for further optimizing the cache design.

This paper is an extended version of our conference paper published at SAMOS [12]. We added significant new contributions to the conference version of the paper. Below is a brief description of the main differences.

- We added Sects. 4.2.2 and 4.2.3 to provide new results for L1/L2 data cache spatial utilization at warp and CTA Levels. The spatial utilization limit study at warp and CTA levels show the contribution of intra-warp and inter-warp accesses to the spatial locality. We believe the quantitative analysis will help to optimize the design of a sector cache. We further improved the results section qualitatively by emphasizing the findings.
- We added Sect. 4.4 to show that the increase in cache locality will increase the performance of memory-divergent workloads.
- We added Sect. 4.5 for potential improvements of GPU cache and warp scheduler design.
- We revised the background section on GPU caches (Sect. 2) and provide a thorough analysis of L1/L2 data caches over 8 generations of NVIDIA architectures along with an overview of GPU memory hierarchy.
- We added Sect. 4.6 to summarize the quantitative analysis, which is very useful for a quick reference to all numbers.

The paper is organized as follows. In Sect. 2, we briefly discuss background on GPU data caches and classify locality. In Sect. 3, we explain our experimental setup. Section 4 presents the quantitative results. Section 5 describes related work. Finally, we conclude in Sect. 6.

2 Background on GPU Cache Organization

The first general-purpose GPU architecture (Tesla architecture) from NVIDIA only had programmer-managed caches. Hardware-managed caches were introduced in GPUs starting from the Fermi architecture. Hardware-managed caches accelerated the use of GPUs for general-purpose computing. Several works compared the performance of Tesla and Fermi GPUs and reported that hardware-managed caches play an important role in the higher performance of Fermi GPUs over Tesla GPUs [3, 7, 29]. GPU caches face different design challenges than CPU caches due to their different characteristics. For instance, write and allocation policies are quite different from CPU caches. The L1 data cache is only write-back for local accesses and write-evict for global accesses whereas in a CPU we either have write-back or write-through caches. The allocation policy is usually no-write allocate for global

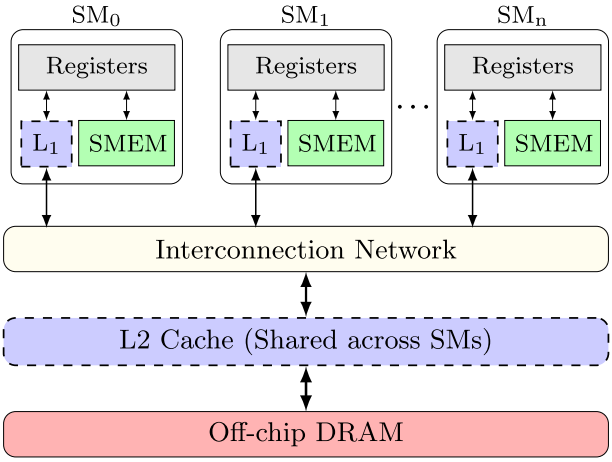


Fig. 1 Overview of a typical memory hierarchy in GPUs

accesses and write-allocate for only local accesses. The deviation in write and allocation policies is to cater to the different requirements of GPU workloads and smaller caches. GPU caches are shared by thousands of threads which make them a scarce resource and a victim of a lot of contention. Furthermore, due to the streaming nature of many GPU applications and smaller cache sizes, caches can suffer from thrashing and a high miss rate. Therefore, exploiting locality is hard due to a large number of active threads and smaller caches. In fact, some works reported negative performance with hardware-managed caches [9, 19].

Figure 1 shows a typical memory hierarchy in GPUs. It consists of a large register file, an on-chip programmer-managed scratchpad, known as shared memory in CUDA, hardware-managed caches (typically two levels), and a high-bandwidth off-chip DRAM, also known as global memory. In addition, GPUs employ constant and texture memories for special purposes.

Table 1 shows L1[1] and L2 data cache size per thread for different generations of NVIDIA architectures. The L1 data cache size ranges from 24 KB to 128 KB, while the L2 data cache size ranges from 768 KB to 6 MB. While both L1 and L2 data cache sizes have increased over the different generations, especially the L2 data cache size, the availability of L1 and L2 data cache per thread is only a few bytes compared to a CPU. For example, Intel Core i9-9900K CPU has 256 KB, 2 MB, 16 MB of L1, L2, and L3 data caches, respectively. With 16 threads this corresponds to 16 KB, 128 KB, and 1 MB of L1, L2, and L3 data cache per thread, respectively. Table 1 shows, in contrast to CPUs, data cache per thread is only 12–64 B for L1 data cache, and for L2 data cache, it is 32–102.4 B. With only a few bytes per thread, it is extremely difficult to exploit spatial and temporal locality for GPUs.

Thread-blocks and warps are thread scheduling granularities in GPUs. Both thread-blocks and warps can affect the data locality in GPU caches. Thread-blocks or co-operative thread arrays (CTA) in NVIDIA terminology are independent units

[1] The largest value is shown for a split between L1 data cache and shared memory.

Table 1 L1 and L2 data cache per thread for NVIDIA's different architectures

Architecture	Representative GPU	Max threads (SM/GPU)	L1 (KB)	L2 (KB)	L1/thread (B)	L2/thread (B)
Tesla	GTX-8800		N/A	N/A	N/A	N/A
Fermi	GTX-580	1536/24576	48	768	32	32
Kepler	GTX-780	2048/24576	48	1536	24	64
Maxwell	GTX-Titan X	2048/49152	24	2048	12	42.6
Pascal	GTX-1080	2048/40960	48	4096	24	102.4
Volta	TITAN V	2048/163840	128	6144	64	38.4
Turing	RTX 2080	1048/50304	64	4096	64	83.4
Ampere	RTX 3080	2048/139264	128	5120	64	37.6

of scheduling on streaming multiprocessors (SMs). A thread block is divided into warps for further scheduling within an SM. A warp consists of 32 threads that execute the same instruction in the lockstep. A thread block can be scheduled on any SM. This feature allows transparent scalability as simply more thread-blocks can be scheduled in parallel when more SMs are available and vice-versa. As thread-blocks can be scheduled on any SM, it is very hard to exploit inter-thread block locality at L1 caches because the L1 cache is private to an SM. For example, when two thread-blocks have inter-thread block locality but they are scheduled on different SMs, there is no way to exploit inter-thread block locality at L1. L2 cache is useful for exploiting inter-thread block locality in this case as the L2 cache is shared among all SMs. Therefore, we have a relatively large L2 cache, which helps to filter requests to off-chip memory. While L1 cache size on GPUs is approaching the size of L1 cache on CPUs and L2 cache is larger than CPUs, the per-thread availability is very low compared to a CPU.

A typical cache line size is 128 B in GPUs. The loads and stores were normally serviced at the granularity of a cache line until the Fermi architecture. However, starting from the Kepler architecture and subsequently, in other architectures such as Maxwell, Pascal, and Volta, the loads and stores can be serviced at 32 B granularity. The 32 B granularity is known as a sector. These architectures still have a cache line size of 128 B, but a cache line is divided into 4 sectors. Such a cache design is also known as a sectored cache. There are byte masks, and on a miss, a sectored cache will only fetch the 32 B sectors that are requested. A full cache line is not automatically fetched, however, if all four sectors are requested, it is also possible to fetch a full cache line.

2.1 Locality Classification

As a thread block (also known as CTA) and a warp, can affect data locality, we classify and study locality at warp and thread block granularities. We classify the locality as *intra-warp locality* when a cache line is initially referenced by a warp and then re-referenced by the same warp. When a cache line is re-referenced by a

different warp than the one initially requested the cache line, we classify such a locality as *inter-warp locality*. Similar to intra-warp and inter-warp locality, we classify the locality as *intra-CTA locality* and *inter-CTA locality*. Table 2 shows a summary of the locality classification.

3 Experimental Setup

3.1 Simulator

We use the cycle-accurate gpgpu-sim simulator for simulating different benchmarks [2]. We configure the simulator to have two-level data caches with a cache line size of 128 B. Table 3 summarizes the main configuration parameters of the simulator.

To generate data for the limit study, we modify the simulator to add counters for L1/L2 cache accesses, warp id, CTA id, bytes accessed within a cache line, etc. We simulate all benchmarks and store counters in a database. The database is quite big (about 1TB) as we store information about all accesses. We then post-process the database using a tool written in C/C++ to extract the locality data presented in the paper.

3.2 Benchmarks

Table 4 shows the benchmarks used for the experimental evaluation. We include benchmarks from the popular Rodinia benchmark suite [4] and CUDA SDK [18] that have memory divergence. From each benchmark, we select a single kernel launch with the highest memory divergence, and when a benchmark has multiple kernels, we only include kernels that have memory divergence. A benchmark is memory divergent if all intra-warp memory accesses of a load or a store instruction cannot be coalesced into one memory transaction/cache access[2]. We use coalescing efficiency to note the degree of memory divergence. The lower the coalescing efficiency, the higher the degree of memory divergence. The coalescing efficiency is given by the following equation:

$$\mathrm{CE} = \sum \mathrm{GMI} / \sum \mathrm{GMT}$$

GMI is the total number of global memory instructions executed for a benchmark and *GMT* is the corresponding number of global memory transactions issued. The non-memory divergent workloads have 100% coalescing efficiency (1/1). The ratio of global memory instructions and global memory transactions is 1 (assuming 32-bit data is accessed by each thread) i.e, for one load from a warp, one memory transaction is generated. In other words, one cache line is accessed for one load from a warp. The full-divergent workloads have about 3% coalescing efficiency (1/32). The ratio of global memory instructions and global memory transactions is 1/32 i.e, for one load from a warp, 32 memory transactions are generated, one transaction for each thread within a warp. In other words, one cache line is accessed for each thread

[2] Two memory transactions/cache accesses when coalescing is done at half warp.

Table 2 Summary of locality classification

Locality category	Description
Intra-warp	Locality from threads of the same warp
Inter-warp	Locality from different warps
Intra-CTA	Locality from warps of the same CTA
Inter-CTA	Locality from different CTAs

Table 3 Summary of simulator configuration

Parameter	Value	Parameter	Value
#SMs	16	Shared memory/SM	48 KB
SM freq (MHz)	822	L1 $ size/SM	16/32 KB
Max #Threads per SM	1536	L2 $ size	768/2304 KB
Max #CTA per SM	8	# Memory controllers	6
Max CTA size	512	Memory type	GDDR5
#FUs per SM	32	Memory clock	2004 MHz
#Registers/SM	32 K	Memory bandwidth	192.4 GB/s

of a warp (assuming 32 threads in a warp). The coalescing efficiency (CE) metric measures the degree of memory divergence, i.e, how many cache accesses for one load/store of a warp. The coalescing efficiency metric is also reported by NVIDIA's profiler. Table 4 shows the coalescing efficiency of the benchmarks.

4 Results

We present the results of the quantitative study of locality in L1 and L2 data caches. In Sect. 4.1, we present results for cache line reuse. Section 4.2.1 presents the analysis of the spatial utilization of cache lines. In Sect. 4.3, we study the hit and miss rate at warp and CTA levels. We calculate the average hit and miss rates by accumulating accesses of all benchmarks.

4.1 GPU Data Cache Lines Reuse Limit

Figure 2 shows the reuse of L1 data cache lines. Figure 2a shows the reuse of L1 data cache lines for 16 KB size. The figure shows that for several kernels such as *histogram*, *kmeans1*, *scan1*, the cache lines are evicted without any reuse. Only a few kernels such as *storeGPU*, *mergeSort2*, and *leukocyte* have all cache lines reused. The average cache line reuse is only 43%, which means that 57% of the cache lines are never re-referenced or get evicted before re-reference.

As GPU caches are much smaller than CPU caches, in particular the share per thread, there is a high probability that the cache lines get evicted before reuse due to high contention. In order to study the lost opportunities to reuse cache lines due to

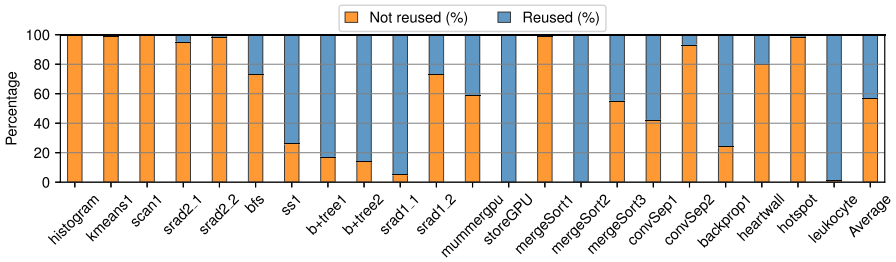
Table 4 Memory-divergent benchmarks used for the experimentation. A benchmark with subscripts designates distinct kernels of the same benchmark

Name	Launch id	CE (%)	Domain	Description
histogram	36	3	Data analytics	Histograms for analysis [18]
kmeans1	1	6	Data mining	k-means clustering [4]
scan1	11	5	Data analytics	Parallel prefix sum [18]
srad2_1	1	56	Image processing	Speckle reducing anisotropic diff. [4]
srad2_2	2	53	Image processing	Speckle reducing anisotropic diff. [4]
bfs	15	47	Graph analytics	Breadth-first search [4]
ss1	9	11	Data mining	Similarity score calculation [4]
b+tree1	1	75	Graph analytics	Graph search [4]
b+tree2	2	75	Graph analytics	Graph search [4]
srad1_1	5	70	Image processing	Speckle reducing anisotropic diff. [4]
srad1_2	11	79	Image processing	Speckle reducing anisotropic diff. [4]
mummergpu	1	7	Bioinformatics	Pairwise local sequence alignment [4]
storeGPU	1	41	Data analytics	Distributed storage systems [1]
mergesort1	2	5	Data analytics	Parallel merge-sort [18]
mergesort2	3	50	Data analytics	Parallel merge-sort [18]
mergesort3	41	67	Data analytics	Parallel merge-sort [18]
convSep1	1	50	Machine learning	Convolution [18]
convSep2	2	43	Machine learning	Convolution [18]
backprop1	2	64	Machine learning	Multi-layer perceptron training [4]
heartwall	2	54	Medical imaging	Ultrasound image tracking [4]
hotspot	1	35	Physics simulation	Processor temperature estimation [4]
leukocyte	3	51	Medical imaging	Microscopy video tracking [4]

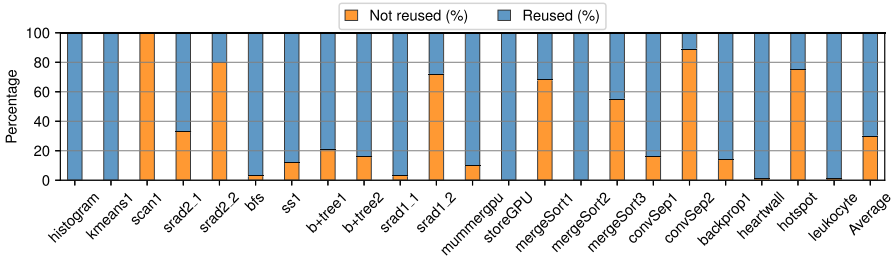
limited cache size, we also simulate an L1 data cache with an infinite size. The infinite cache size here implies that once a cache line is brought to the cache, it never gets evicted. Figure 2b shows the L1 data cache lines reuse with an infinite size. The average cache lines reuse for the infinite L1 data cache is 70%, which shows that most of the kernels have high locality but current GPUs are unable to exploit the locality due to limited cache size. However, even with the infinite cache size, 30% of the L1 data cache lines have no data reuse. These cache lines can be safely replaced or evicted without any loss in data locality.

The L1 data cache lines analysis shows low reuse, especially for the default cache size. Along with the reuse, it is also important that the cache lines are spatially utilized, otherwise, we might be wasting significant cache capacity for unneeded data. Therefore, in the next section, we will study the spatial utilization of cache lines in detail.

Observation 1: The limit study of cache lines reuse shows a much higher reuse (70%) than currently exploited (43%) by GPUs.



(a) L1 data cache lines reuse with 16 KB size.



(b) Limit of L1 data cache lines reuse with with an infinite size.

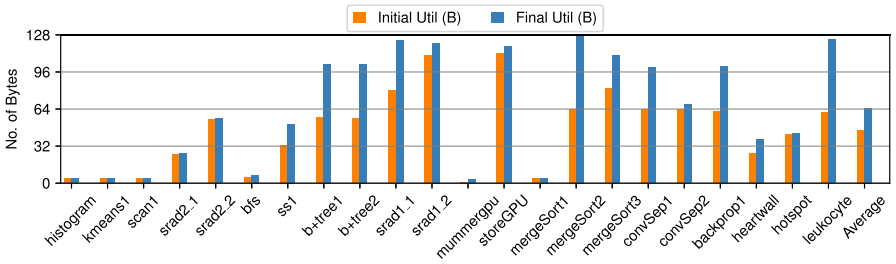
Fig. 2 GPU L1 data cache lines reuse

4.2 Spatial Utilization of GPU Data Cache Lines

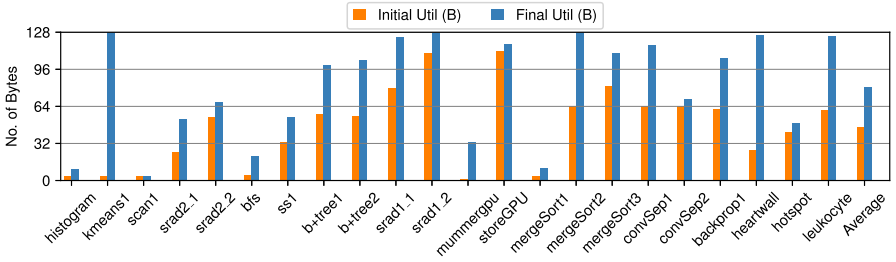
GPUs use a wide cache line size as threads from a warp usually access nearby data that can be packed into one cache line. This is true for non-memory divergent applications, however, for memory-divergent applications wider cache lines result in severe wastage of resources as most of the bytes are left unused [20]. We study in-depth the spatial utilization of cache lines. While some of the findings are intuitive, there is a lack of quantitative numbers to support the intuition. Our study fulfills this gap by providing concrete numbers for the spatial utilization of cache lines, enabling optimizations for the cache design.

4.2.1 Spatial Utilization Limit and Over-fetch

Figure 3 shows the spatial utilization of L1 data cache lines. Each kernel has two bars. The first bar shows the average initial utilization of a cache line, i.e. when a cache line is initially fetched. The second bar shows the average final utilization of a cache line, i.e., when a cache line is evicted. For calculating the spatial utilization, we use a bit vector of length equal to a cache line size. The bits corresponding to bytes that are requested at the time of fetching a cache line are initially set, and the bit vector is updated whenever a cache line is reused until the cache line is finally evicted.



(a) L1 data cache lines spatial utilization with 16 KB size.



(b) Limit of L1 data cache lines spatial utilization with an infinite size.

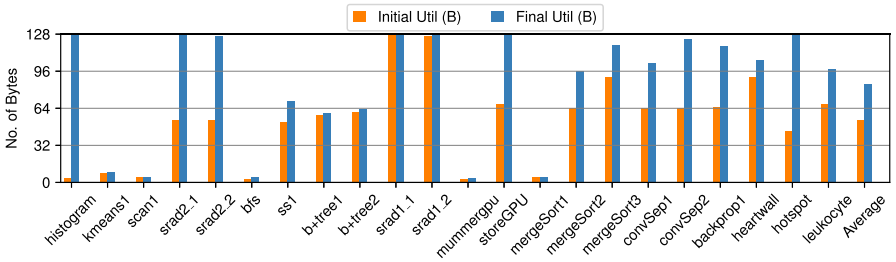
Fig. 3 GPU L1 data cache lines spatial utilization showing data over-fetch

Figure 3a shows the spatial utilization of the L1 data cache lines for 16 KB cache size. The figure shows that for many kernels, the initial and the final utilization of cache lines is almost the same, which means there is a very low spatial utilization of data cache lines like reuse. Only kernels such as *b+tree1*, *b+tree2*, *srad1*, *mergesort2*, *mergeSort3*, *convSep1*, *backprop1*, and *leukocyte* have significantly higher final spatial utilization of cache lines. The average initial and final utilization of cache lines is 46 B and 65 B for all kernels, meaning 63 B on average are over-fetched and never used. This implies that about 50% of the cache capacity (63 B out of 128 B of a cache line), and other scarce resources such as network-on-chip bandwidth, L2 data cache bandwidth, etc., are wasted due to the data over-fetch.

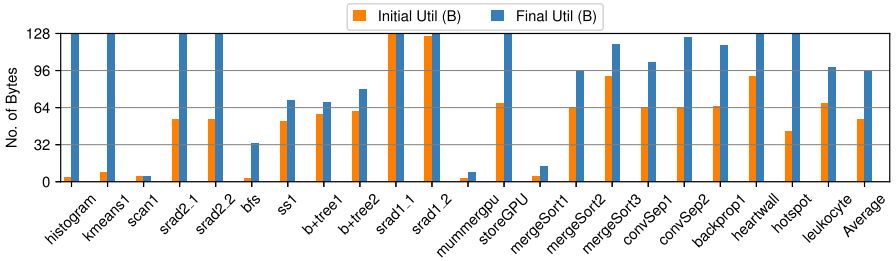
The low spatial utilization of cache lines could also be due to the limited cache capacity and high contention as a cache line may be evicted before it gets re-referenced. To study the upper limit of the spatial utilization of cache lines, we simulate an infinite L1 data cache. Figure 3b shows the final utilization of cache lines for an infinite L1 data cache. The figure shows that most of the kernels actually have higher spatial locality than exploited by the default cache size, however, GPUs are unable to exploit full spatial locality. The average final utilization of cache lines for the infinite L1 data cache is 81 B, which is 24% higher than the default cache size. However, even with the infinite cache size, 36% of the cache capacity is still wasted. This implies that there is a potential for further improvements in the cache design.

There are two important conclusions that we can draw from the spatial utilization of cache lines. First, on the one hand, the low initial spatial utilization of cache lines justifies the sector cache design of new GPU architectures such as Maxwell, Pascal, and Volta architectures. That is only fetch the sectors that are required at the time of issuing a memory request. On the other hand, the much higher final spatial utilization of cache lines, as shown by our experiments, presents an opportunity for further optimizing cache designs. For example, if we only fetch the sectors on demand depending on the initial utilization of a cache line, we significantly lose the spatial locality present in the kernels. The initial sectors that need to be fetched for a memory request can be determined at the time of coalescing memory requests of threads of a warp. If we only fetch the sectors as determined during coalescing, which is how it is done in recent GPUs, there will be significant lost opportunities for exploiting locality as our analysis shows that the final utilization of cache lines is much higher than the initial utilization of cache lines (see Fig. 3). One possible idea to exploit the lost locality is to investigate the use of spatial locality predictor and depending upon the prediction, fetch the predicted sectors.

We also did a similar kind of analysis for the L2 data cache. Figure 4 shows the spatial utilization of L2 data cache lines for 128 KB as well as infinite cache sizes. Figure 4 shows that the L2 data cache has better spatial utilization of cache lines compared to the L1 data cache. This is because the L2 data cache is shared by multiple processors. The average initial and the final utilization of cache lines is 54 B and 85 B, respectively. This means that 35% of the L2 data cache capacity is



(a) L2 data cache lines spatial utilization with 128 KB size.



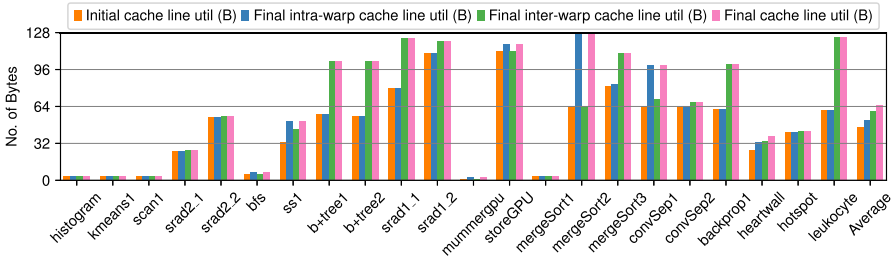
(b) Limit of L2 data cache lines spatial utilization with an infinite size.

Fig. 4 GPU L2 data cache lines spatial utilization showing data over-fetch

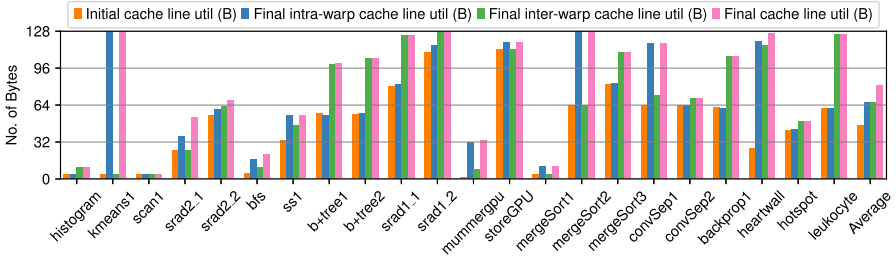
wasted, which is lower than the L1 data cache but still significant to look for further optimizations. The wastage of the L2 data cache is also a measure of wasted off-chip memory bandwidth. The average final utilization of cache lines is 95 B for the infinite cache, which is almost 2.0× more compared to the average initial utilization of cache lines. The big difference between the initial and the final utilization of cache lines shows that pure demand-based sector fetching, implemented in GPU caches, needs to be revisited to improve their performance. The analysis also shows that we need more aggressive policies to properly utilize the L1 cache than the L2 cache.

Observation 2: The spatial utilization study shows that about 50% of the L1 cache capacity, and other scarce resources such as NoC bandwidth, memory bandwidth are wasted due to data over-fetch. The limit study shows 64% spatial utilization, warranting further optimizations to cache designs. For instance, the pure demand-based sector fetching in GPU caches can be improved by using a simple spatial locality predictor.

We studied the spatial utilization of cache lines and their limit (Figs. 3 and 4). However, the study does not show how much spatial locality is exploited at a warp and a CTA level. Because a warp and a CTA are two important units of scheduling in GPUs, and both of them can affect the spatial utilization of cache lines, we also study in detail the spatial locality at warp and CTA levels in the next section.



(a) Intra- and inter-warp spatial utilization of L1 data cache with 16 KB size.



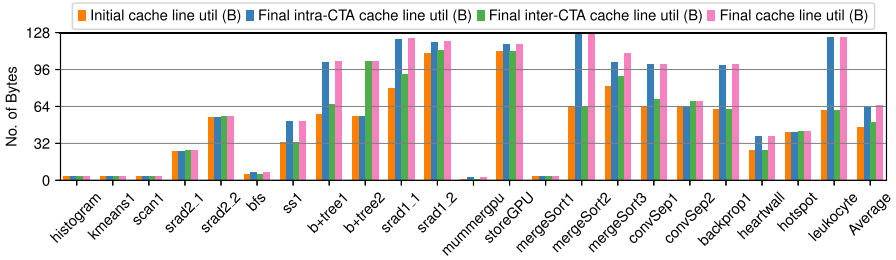
(b) Intra- and inter-warp spatial utilization of L1 data cache with an infinite size.

Fig. 5 Intra- and inter-warp spatial utilization of L1 data cache

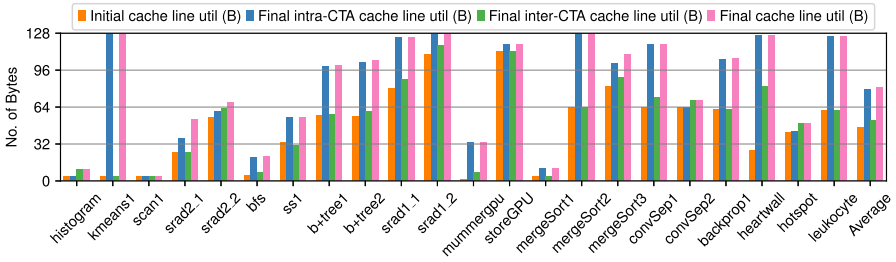
4.2.2 L1 Data Cache Spatial Utilization at Warp and CTA Levels

Figure 5 shows the intra- and inter-warp spatial utilization of the L1 data cache. Figure shows the initial cache line utilization, that is the number of bytes needed at the time of a cache line allocation, final cache line utilization, the number of bytes utilized when the cache line is evicted. The figure also shows the final intra-warp cache line utilization, which is the number of bytes utilized at the time of eviction by the same warp that initially allocated the cache line. Similarly, the final inter-warp cache line utilization shows the number of bytes used at the time of eviction by different warps than the one which allocated the cache line. The final cache line utilization is the utilization by all warps, without differentiating between intra- and inter-warps, at the time of eviction.

Figure 5a shows the warp level spatial locality for the 16 KB cache. The average final intra- and inter-warp spatial utilization is 52 B and 60 B for the 16 KB cache, while for the infinite cache, the final intra- and inter-warp spatial utilization is 66 B. Figure 6 shows the final intra- and inter-CTA spatial utilization of L1 data cache for both 16 KB and infinite cache. The final intra-CTA cache line utilization shows the number of bytes used by all warps in a CTA when the cache line is evicted. Similarly, the final inter-CTA cache line utilization shows the number of bytes used by other CTAs than the one which allocated the cache line when the cache line is evicted.



(a) Intra- and inter-CTA spatial utilization of L1 data cache with 16 KB size.



(b) Intra- and inter-CTA spatial utilization of L1 data cache with an infinite size.

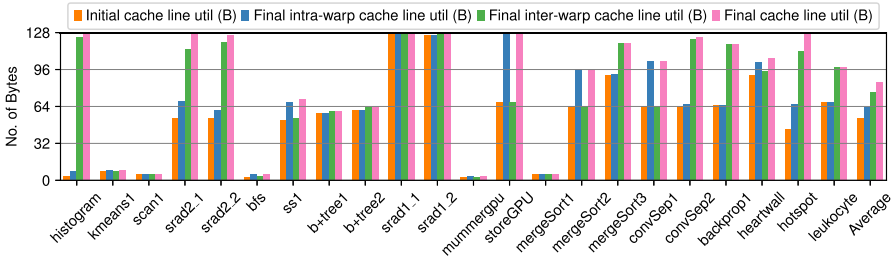
Fig. 6 Intra- and inter-CTA spatial utilization of L1 data cache

While the warp level spatial locality (Fig. 5) shows that intra- and inter-warp accesses contribute almost the same to the spatial locality, the CTA level analysis shows that intra-CTA (79 B) accesses contribute much more spatial locality than the inter-CTA (52 B). The higher final intra-CTA spatial utilization also means that the warps within a CTA contribute higher to spatial locality, and they should be prioritized for scheduling over warps from a different CTA. While Fig. 5 shows that inter-warp locality is close to intra-warp, it was not clear that these inter-warps belong to the same CTA.

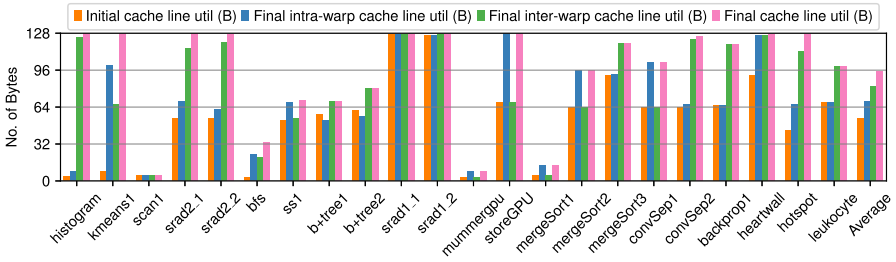
Observation 3: The spatial utilization limit study at warp and CTA levels shows that intra-warp and inter-warp accesses contribute equally to the spatial locality, while most of the inter-warp locality belongs to the warps from intra-CTAs rather than inter-CTAs (79 B vs. 52 B). This implies that optimizations targeting spatial utilization such as spatial locality predictor should consider both intra-warp and inter-warp accesses from the same CTA as an important feature.

4.2.3 L2 Data Cache Spatial Utilization at Warp and CTA Levels

Figures 7 and 8 show the warp and CTA levels spatial utilization of L2 data cache, respectively. The figures show that the spatial utilization of L2 data cache is higher than L1 data cache owing to its shared nature across all streaming multiprocessors. The final intra- and inter-warp spatial utilization is 63 B and 76 B for the 128 KB L2 data cache, while for the infinite cache, the final intra- and inter-warp spatial

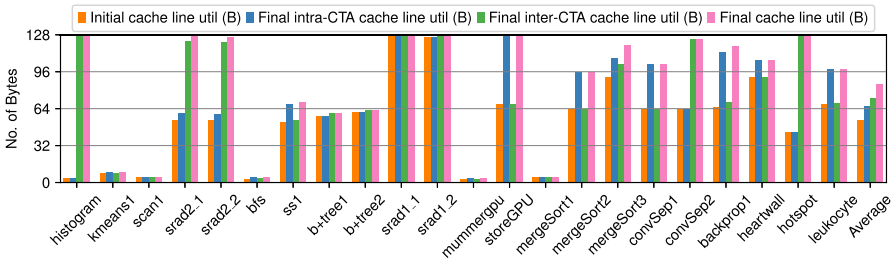


(a) Intra- and inter-warp spatial utilization of L2 data cache with 128 KB size.

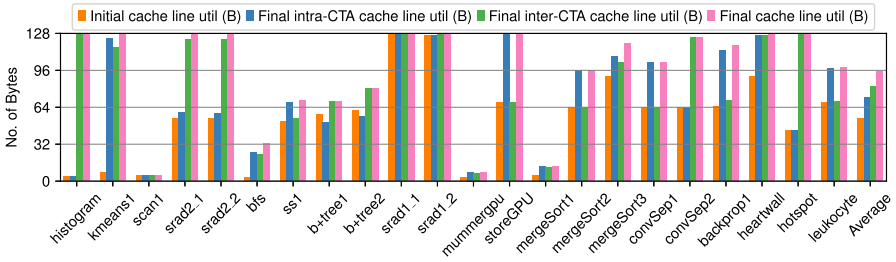


(b) Intra- and inter-warp spatial utilization of L2 data cache with an infinite size.

Fig. 7 Intra- and inter-warp spatial utilization of L2 data cache



(a) Intra- and inter-CTA spatial utilization of L2 data cache with 128 KB size.



(b) Intra- and inter-CTA spatial utilization of L2 data cache with an infinite size.

Fig. 8 Intra- and inter-CTA spatial utilization of L2 data cache

utilization is 69 B and 82 B, respectively. The final intra- and inter-CTA spatial utilization is 66 B and 73 B for the 128 KB L2 data cache, while for the infinite cache intra- and inter-CTA spatial utilization is 73 B and 82 B, respectively. The final L2 data cache lines utilization is 85 B and 95 B for 128 KB and the infinite size, respectively.

While the results show higher inter-warp locality both for the finite and infinite L2 data cache (Fig. 7), there are some interesting observations from the CTA level results as shown in Fig. 8. Figure 7 shows that inter-warp contributes more to data locality, however, unlike the L1 data cache, these inter-warps belong to different CTAs as shown by higher utilization in Fig. 8.

Observation 4: The spatial utilization limit study at warp and CTA levels shows that inter-warp accesses contribute more to the spatial locality (82 B vs. 69 B), while most of the inter-warp locality comes from warps that belong to inter-CTAs rather than intra-CTAs (82 B vs. 73 B), unlike the L1 data cache.

4.3 Hit and Miss Rate Limit

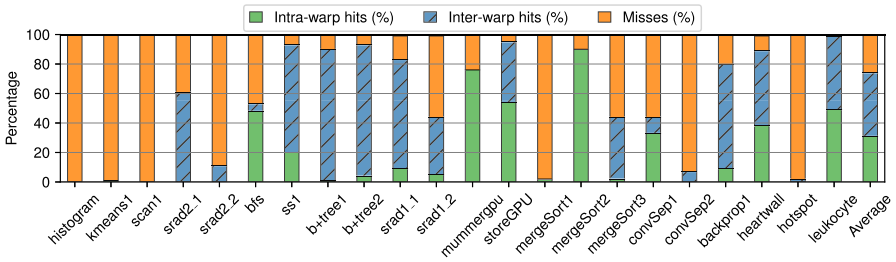
As hit and miss rates are important metrics to evaluate a cache design, we also present hits and misses for both L1 and L2 data caches. We classify hits into four categories, *intra-warp* and *inter-warp hits*, *intra-CTA* and *inter-CTA hits*. As a warp is an important scheduling unit within a streaming multiprocessor, the hit ratio at a

warp granularity provides crucial information that could be used for both scheduling warps and improving cache design.

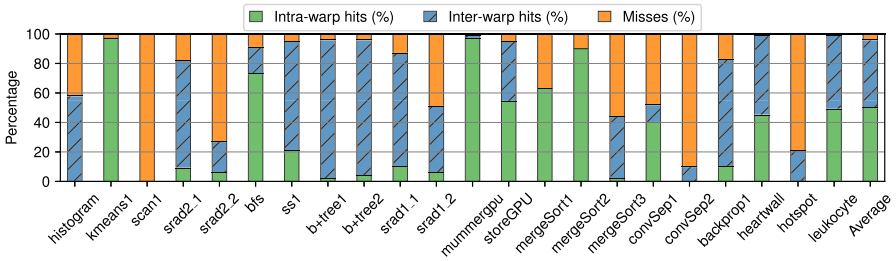
4.3.1 Warp-Level Hit and Miss Rate

Figure 9 shows the intra- and inter-warp hits of the L1 data cache. Figure 9a shows that the L1 data cache has 31% intra-warp and 43% inter-warp hits for 16 KB cache size. The miss rate is 26% which is quite high. In contrast to GPUs, CPUs have a hit rate typically between 95 and 97% for an L1 cache [6]. As discussed earlier, it is extremely difficult to exploit data locality in GPU caches due to several factors such as small size, and massive multithreading.

Figure 9b shows the intra- and inter-warp hits for the L1 data cache with the infinite size. The figure shows that the average hit rate is about 96%, which is close to a typical hit rate for a CPU L1 cache. Concretely, on average, we have 50% intra-warp and 46% inter-warp hits. There are a couple of interesting observations. First, kernels actually have a significantly higher locality but the current GPUs are not able to fully exploit it. Second, in contrast to some existing work [21], which shows much higher intra-warp locality than inter-warp locality, we see an almost equal division between the intra-warp and inter-warp localities. This is possible for memory-divergent workloads as they have scattered memory accesses and threads from different warps (which contribute to inter-warp hits) can access the data from



(a) Intra- and inter-warp locality of L1 data cache with 16 KB size.



(b) Intra- and inter-warp locality of L1 data cache with an infinite size.

Fig. 9 Intra- and inter-warp locality of L1 data cache

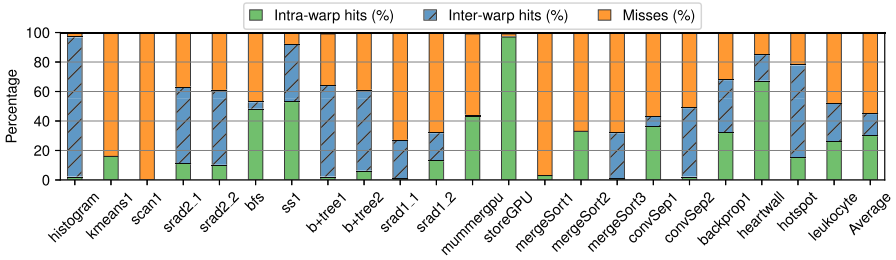
the same cache line. This can happen more frequently for memory-divergent workloads.

We also study the intra- and inter-warp hits for the L2 data cache. Figure 10 shows the intra- and inter-warp hits for the L2 data cache. The average intra-warp and inter-warp hits are 30% and 15%, respectively, for the 128 KB size, while for the infinite size, the average intra-warp and inter-warp hits are 65% and 21%, respectively. An important point that is not evident after looking at the intra- and inter-warp locality is whether the warps belong to the same CTA or different CTAs. This could also provide useful insight as CTAs are units of work distribution across different streaming multiprocessors. Moreover, the quantitative numbers further provide information on making scheduling decisions, for example, should we first schedule the warps from the same CTA or different CTAs.

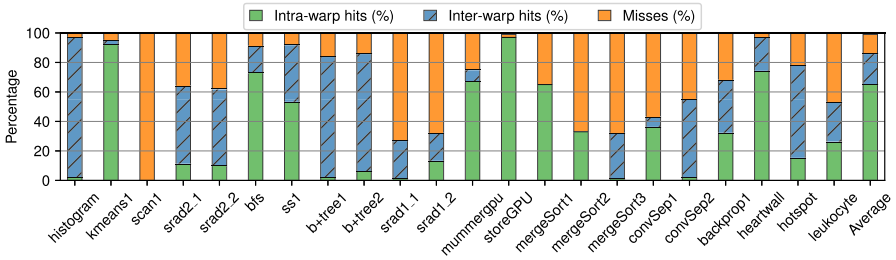
Observation 5: The limit study shows significantly higher inter-warp hits (46%) at the L1 cache in contrast to the previous study [21]. This is possible for memory-divergent workloads as they have scattered memory accesses.

4.3.2 CTA-Level Hit and Miss Rate

Figure 11 shows the intra- and inter-CTA locality of the L1 data cache for 16 KB and infinite sizes. Figure 11a shows that on average, there are 68% intra-CTA hits and 5% inter-CTA hits, which shows that a high percentage of inter-warp hits shown in Fig. 9 are basically from warps within the same CTA. This is more evident when

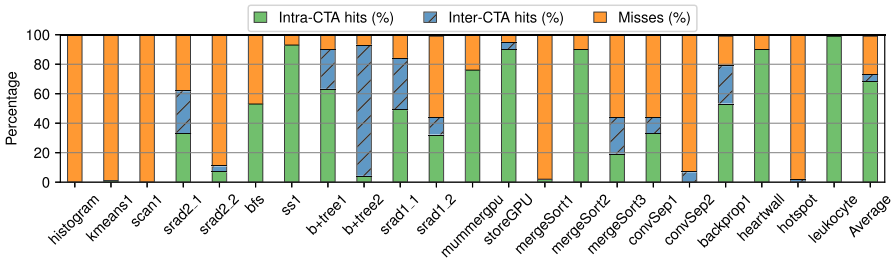


(a) Intra- and inter-warp locality of L2 data cache with 128 KB size.

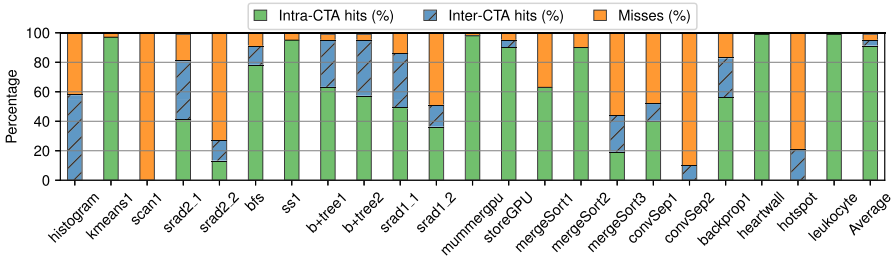


(b) Intra- and inter-warp locality of L2 data cache with an infinite size.

Fig. 10 Intra- and inter-warp locality of L2 data cache



(a) Intra- and inter-CTA locality of L1 data cache with 16 KB size.



(b) Intra- and inter-CTA locality of L1 data cache with an infinite size.

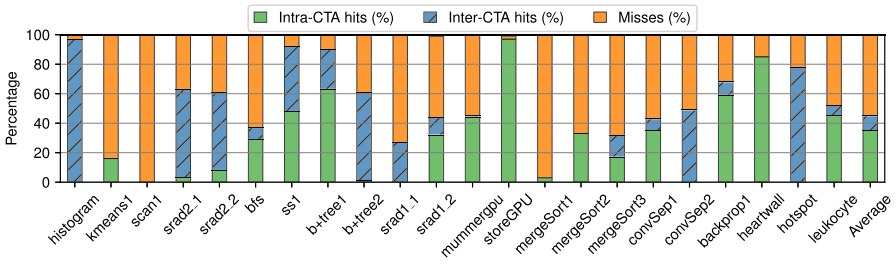
Fig. 11 Intra- and inter-CTA locality of L1 data cache

we look at the average numbers for the infinite L1 data cache. On average, 91% hits are intra-CTA and only 5% are inter-CTA. Figure 12 shows a similar analysis for the L2 data cache. For the infinite L2 data cache, inter-CTA hits are 13%, which shows that there is a potential for better scheduling. For example, if we can schedule such CTAs on the same streaming multiprocessor, there is a possibility to exploit more inter-CTA locality at the L1 data cache that can lead to better performance.

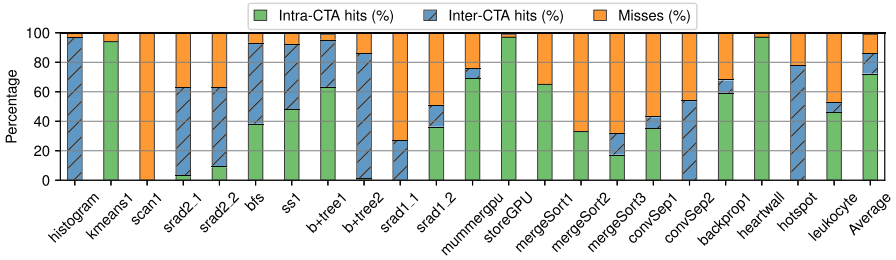
Observation 6: The CTA level hit rate confirms a higher intra-CTA locality at the L1 cache and inter-CTA locality at the L2 cache.

4.4 Cache Sensitivity of Memory-Divergent Workloads

We show that memory-divergent workloads have much higher data locality than currently exploited by GPUs (see Sects. 4.1, 4.2, 4.3). To further show that the increase in data locality will boost the performance of memory-divergent workloads instead of degrading them, we conduct experiments by increasing the L1 cache size. Although, the increase in the cache size is not a direct measure of increased locality, it nonetheless, helps to capture the locality that otherwise gets lost due to smaller cache size in GPUs. Figure 13 shows the speedup of memory-divergent benchmarks with the increase of L1 cache size. The results show that the performance of memory-divergent benchmarks increases up to 6.3× and on average by 1.62×. More importantly, the performance does not degrade, not even in one case. Although not all benchmarks gain in performance, the increase in data locality will



(a) Intra- and inter-CTA locality of L2 data cache with 128 KB size.



(b) Intra- and inter-CTA locality of L2 data cache with an infinite size.

Fig. 12 Intra- and inter-CTA locality of L2 data cache

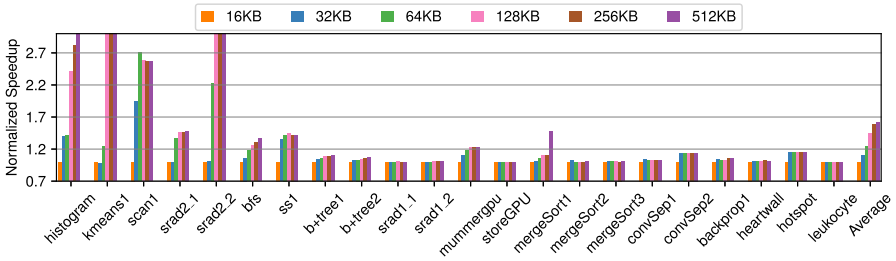


Fig. 13 Effect of L1 size on the performance of memory-divergent workloads

save resources such as cache capacity, memory bandwidth, which can be used by other concurrently running benchmarks, or to save power consumption.

4.5 Potential Improvements to GPU Cache and Warp Scheduler Design

Based on the quantitative analysis of data locality for memory-divergent workloads, we see two main architectural improvements that can potentially improve the performance of GPUs.

4.5.1 Augment Sector Cache with Spatial Locality Prediction

Based on the findings of the limit study that GPU L1/L2 caches have much higher spatial locality than currently exploited by GPUs, we see an excellent opportunity to improve the spatial utilization of caches (and performance of GPUs) by fetching sectors into a cache based on a spatial locality predictor. The current generations of GPUs employ sector caches and fetch only those sectors of a cache line that are required at the time of issuing a memory request. While a sector cache design can save significant over fetching of data compared to a non-sector cache, it is still a conservative design and misses the opportunities to exploit higher spatial locality present in GPU caches. Our limit study shows that the final spatial utilization of L1 cache lines (81 B) is 24% higher than the default cache size (65 B) and 76% higher than the initial spatial utilization (46 B). If we only fetch sectors based on the initial utilization, which is how it is done in current GPUs, there will be significant lost opportunities for exploiting locality. Moreover, we need to issue additional memory requests to fetch the missing sectors of the same cache line.

A table-based spatial locality prediction has been proposed to improve the cache design of CPUs [10]. While it will be interesting to explore the possibility of adopting a table-based spatial locality predictor also for GPUs, the more recent approaches such as using deep learning to improve GPU architecture might be more accurate and performance efficient [24]. In any case, our limit study at warp and CTA levels further help to improve the design of the predictor by providing significant insights about the entries that should be cached in a table-based design or the features that should be explored in a deep learning-based predictor.

4.5.2 Warp Scheduler Improvements

GPU warp scheduling is a well-explored area with different schedulers typically taking warp and CTA levels locality into consideration while making scheduling decisions [17, 21, 22]. For example, Rogers et al. [21] show much higher intra-warp locality than inter-warp locality, hence, the proposed warp scheduler also gives priority to intra-warp accesses. In contrast to the previous work, which did not focus on memory-divergent workloads, our limit study shows 50% intra-warp and 46% inter-warp hits at the L1 cache for memory-divergent workloads. This is possible for memory-divergent workloads as they have scattered memory accesses and threads from different warps (which contribute to inter-warp hits) can access the data from the same cache line. This can happen more frequently for memory-divergent workloads compared to regular workloads. Therefore, we see another possibility to improve the warp/CTA scheduling decisions by taking the memory-divergence of an application into account. A simple memory access pattern sampler could be employed to figure out the type of workloads, memory-divergent vs. non-memory divergent, indicating the difference in the data locality of workloads. A locality-driven warp scheduler can adjust the scheduling decisions as per the sampler information. As the main goal of this work is to study the data locality of memory-divergent workloads in detail, we leave the potential architectural improvements for the future.

4.6 Summary of Quantitative Analysis

Table 5 presents a summary of the quantitative analysis of data locality in GPU data caches for quick reference. Table shows cache lines reuse, spatial utilization, and hit rate at warp and CTA level granularities. We only show the final numbers with default and infinite cache sizes for L1 and L2 data caches for brevity. We assume a cache line size of 128 B. The spatial utilization numbers are in bytes. The unused bytes for a cache line can be simply calculated from the table. For example, a spatial utilization of 65 B with 16 KB of L1 cache means that 63 B are left unused. Similarly, the percentage of misses is not shown but can also be calculated from the table. For example, the intra- and inter-warp hits are 31% and 43% for 16 KB L1 data cache, which means the misses are 26%.

5 Related Work

We divide the related work into two categories. First, we present the related work on the study of locality in GPU caches. As we use memory-divergent workloads, we also present the main work done on memory divergence. We discuss the work done to tolerate memory divergence as well as work performed to reduce/eliminate memory divergence.

5.1 Related Work on Locality in GPU Caches

Several works report that hardware-managed caches play an important role in the higher performance of GPUs [3, 7, 29]. It is also well-known that exploiting locality is crucial in GPUs, otherwise, hardware-managed caches can also cause negative performance [9, 19, 27]. As sometimes, the use of caches can degrade performance, there have been several studies, which show that bypassing caches could lead to higher performance [5, 13, 15, 30]. However, to the best of our knowledge, there is not much work done to quantify the locality in GPU caches. Rogers et al. [21]

Table 5 Summary of the quantitative analysis of locality in GPU data caches

Parameter	L1 data cache		L2 data cache	
	16 KB	Infinite	128 KB	Infinite
Reuse	43%	70%	–	–
Spatial util	65 B	81 B	85 B	95 B
Intra-warp util	52 B	66 B	63 B	69 B
Inter-warp util	60 B	66 B	76 B	82 B
Intra-CTA util	65 B	81 B	66 B	73 B
Inter-CTA util	65 B	81 B	73 B	82 B
Intra-warp hits	31%	50%	30%	65%
Inter-warp hits	43%	46%	15%	21%
Intra-CTA hits	68%	91%	35%	72%
Inter-CTA hits	5%	5%	10%	14%

propose a cache-aware warp scheduling mechanism, based on estimated intra-warp locality, to capture locality that is lost by other schedulers due to excessive contention for cache capacity. They reported that the majority of data reuse observed in highly cache sensitive benchmarks comes from the intra-warp locality and therefore, give priority to intra-warp instructions to access the L1 data cache. Li et al. [14] quantify the percentage of the inter-CTA reuse, which is based on the data reuse of all the memory requests generated from streaming multiprocessors before they enter the L1 cache. Tang et al. [25] quantitatively analyze the data reuse of dynamic applications and propose hardware scheduler to improve data locality at runtime. In contrast, we present a comprehensive quantitative study of L1 and L2 data caches in GPUs, highlighting the gap between the locality exploited and the maximum locality actually present in memory-divergent workloads.

5.2 Related Work on Memory Divergence

Meng et al. [16] introduce dynamic warp subdivision (DWS), which allows a single warp to occupy more than one slot in the scheduler without requiring extra register file space. Independent scheduling entities allow divergent branch paths to interleave their execution, and allow threads that hit to run ahead. The DWS does not improve memory divergence, but results in improved latency hiding and memory level parallelism. Tarjan et al. [26] propose adaptive slip, which allows a subset of threads from SIMD warps to continue execution while other threads in the same warp are waiting for memory. This provides benefits when runahead threads prefetch cache lines for lagging threads. It also increases throughput when divergent threads experience misses and runahead and lagging threads continually leapfrog each other, rather than continually being held back by the slowest thread. The key insight is that SIMD cores' support for branch divergence can be elegantly extended to support memory divergence.

Zhang et al. [31] propose a software solution called G-Streamline to improve both irregular memory references and control flow. G-Streamline eliminates irregularity by enhancing the thread-data mappings on the fly. The key idea is that both irregular memory references and control flows essentially stem from an inferior mapping between threads and data (data locations for the former; data values for the latter). This perspective leads to the basic strategy of G-Streamline for irregularity elimination: enhancing the thread-data mappings on the fly. Dynamic irregular references are those whose memory access patterns are unknown (or hard to know) until the execution time. Being dynamic, these references are especially hard to tackle, making effective exploitation of GPUs difficult for many applications in various domains, including fluid simulation, image reconstruction, dynamic programming, data mining.

Sartori and Kumar [23] propose branch and data herding, which exploit error tolerance of applications to reduce memory divergence. Branch and data herding is based on the observation that many GPU applications produce acceptable outputs even if a small number of threads in a SIMD execution unit are forced to go down the wrong control path or are forced to load from an incorrect address. Data herding eliminates memory divergence by forcing each thread in a warp to load from the

most popular memory block accessed by a warp load, allowing all the loads to be coalesced into a single memory request. However, there is an obvious limitation that it can only be used for error-tolerant applications. Lal et al. [11] investigate performance bottlenecks in GPUs and show that several workloads have a high degree of memory divergence that can cause low performance.

In addition to memory divergence tolerance, reduction, and elimination, some recent studies have also focused to model the performance of memory divergence workloads [8, 28]. Huang et al. [8] propose the GPUMech, an analytical performance model for GPU architectures based on interval analysis. Wang et al. [28] also propose an analytical performance model for GPUs called, the Memory Divergence Model (MDM). While both models are equally accurate for non-memory divergent applications, MDM achieves higher accuracy (13.9% average prediction error) compared to GPUMech (162% average prediction error) for memory-divergent applications.

6 Conclusions

GPUs can deliver peak performance in TFLOPs, however, the peak performance is often difficult to achieve due to several performance bottlenecks. Memory divergence is one such performance bottleneck that can lead to poor cache performance such as cache thrashing and high miss rate, impeding GPU performance. As exploiting data locality is crucial for performance, we conduct the first detailed quantitative study of data locality in GPU caches, showing the limits of locality for memory-divergent workloads at different granularities. Our analysis shows that memory-divergent workloads have significantly higher locality, but the current GPUs are not able to fully exploit it. We show that 57% of the cache lines are never re-referenced, however, the limit study shows that only 30% of the cache lines are never re-referenced and 27% are evicted before re-reference. We further find that about 50% of the L1 data cache capacity, consequently also NoC bandwidth, are wasted due to data over-fetch. While the low spatial utilization justifies the sectored-cache design as it can save some scarce resources, the limit study reveals that the sectored-cache design is conservative and misses the opportunities to exploit higher spatial locality present in GPU caches. We also study the temporal and spatial locality at warp and CTA levels and observe that, in contrast to previous studies, there is a significantly higher inter-warp locality for L1 data cache for memory-divergent workloads due to scattered accesses, which can influence warp scheduling policies. Similarly, the locality analysis at the CTA level shows 13% inter-CTA hits at the L2 data cache, which shows the potential for better CTA scheduling across multiprocessors. While some of the findings are intuitive, there is a lack of data, which our study fulfills by providing concrete numbers. Based on the quantitative analysis, we also provide some suggestions to improve the cache design. We believe that our findings provide significant insights that will help to improve the GPU cache design. In the future, we plan to use some of the key insights to improve GPU performance.

Appendix

Analysis of L1 Data Cache Locality with 32 KB Size

We conducted new experiments with increased L1 and L2 data cache sizes to match recent GPUs. As shown in Table 1 in the main paper (L1 and L2 data cache per thread for NVIDIA's different architectures), Volta/Turing/Ampere architectures have 2× more L1 data cache per thread compared to the Fermi architecture. Although, Volta and Ampere architectures have only a bit more share of L2 data cache per thread, Turing has about 2.6× more L2 data cache per thread. Therefore, we increased the L1 data cache size by 2× (16 × 2 = 32 KB) and L2 data cache size by 3× (128 × 3 = 384 KB) to be representative of more recent architectures and repeated experiments for data locality analysis. We only present the main results for L1/L2 data caches as we only observe small changes with respect to results presented in the main paper.

Figures 14, 15, 16, and 17 show the L1 data cache lines reuse, spatial utilization, intra- and inter-warp spatial utilization, and intra- and inter-warp hits for 32 KB size. As expected, we only see small improvements in the L1 data cache lines reuse (3.6%), spatial utilization (4%), and hit rate (1.1%).

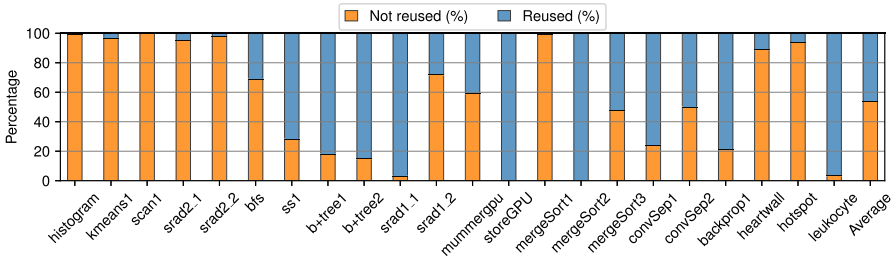


Fig. 14 L1 data cache lines reuse with 32 KB size

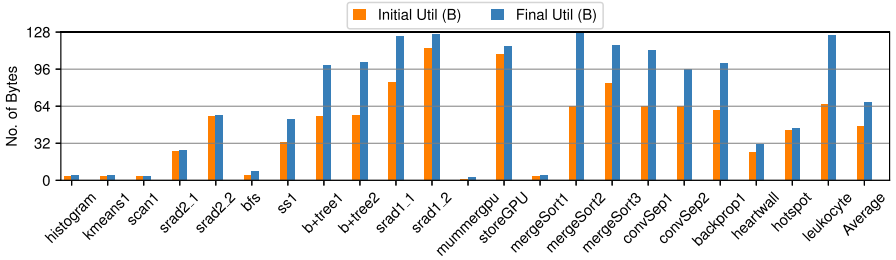


Fig. 15 L1 data cache lines spatial utilization with 32 KB size

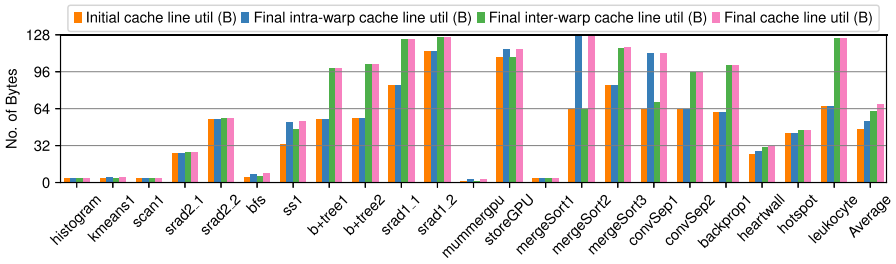


Fig. 16 Intra- and inter-warp spatial utilization of L1 data cache with 32 KB size

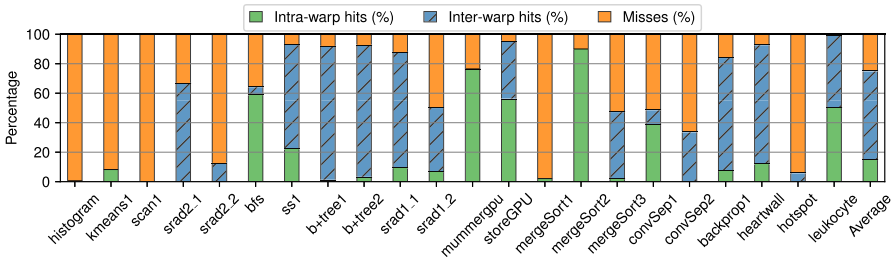


Fig. 17 Intra- and inter-warp locality of L1 data cache with 32 KB size

Analysis of L2 Data Cache Locality with 384 KB Size

Figures 18, 19 and 20 show the L2 data cache lines spatial utilization, intra- and inter-warp spatial utilization, and intra- and inter-warp hits for 384 KB size. Similar to the L1 data cache, we also see small improvements in the L2 data cache lines spatial utilization (3%), and hit rate (18%) when the L2 cache size is increased by 3×. This shows that more recent GPUs will also have similar issues. Therefore, the potential improvements as suggested in the paper are also applicable for more recent GPUs.

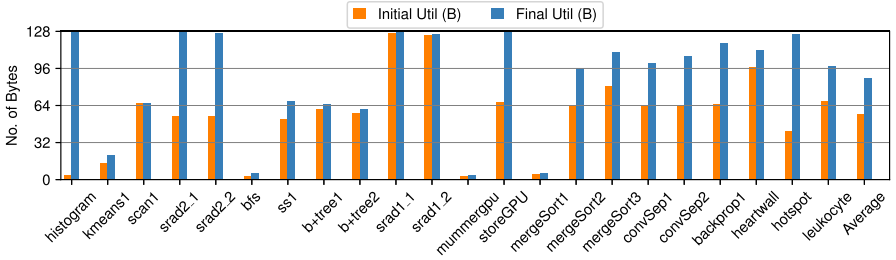


Fig. 18 L2 data cache lines spatial utilization with 384 KB size

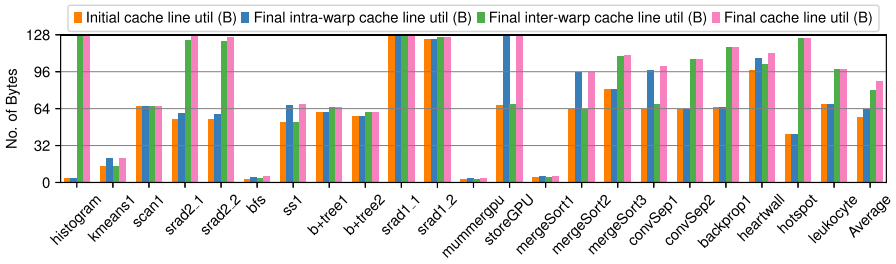


Fig. 19 Intra- and inter-warp spatial utilization of L2 data cache with 384 KB size

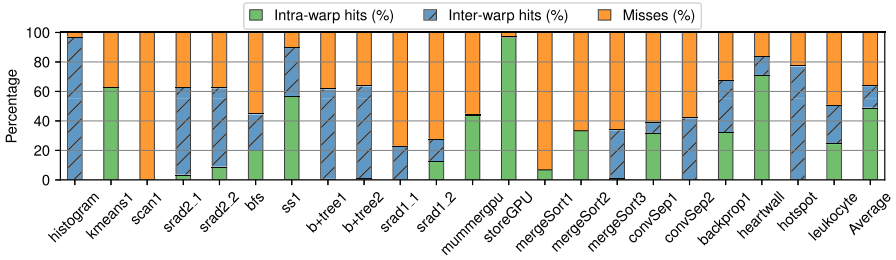


Fig. 20 Intra- and inter-warp locality of L2 data cache with 384 KB size

Funding Open Access funding enabled and organized by Projekt DEAL.

References

1. Al-Kiswany, S., Gharaibeh, A., Santos-Neto, E., Yuan, G., Ripeanu, M.: StoreGPU: Exploiting graphics processing units to accelerate distributed storage systems. In: Proceedings of the 17th International symposium on high performance distributed computing, HPDC (2008)
2. Bakhoda, A., Yuan, G.L., Fung, W.W.L., Wong, H., Aamodt, T.M.: Analyzing CUDA workloads using a detailed GPU simulator. In: Proceedings of the IEEE international symposium on performance analysis of systems and software, ISPASS, https://github.com/gpgpu-sim/ (2009)
3. Cederman, D., Chatterjee, B., Tsigas, P.: Understanding the performance of concurrent data structures on graphics processors. In: Proceedings of the 18th international conference on parallel processing, Euro-Par (2012)
4. Che, S., Boyer, M., Meng, J., Tarjan, D., Sheaffer, J., Lee, S.H., Skadron, K.: Rodinia: A benchmark suite for heterogeneous computing. In: Proceedings of the IEEE International Symposium on Workload Characterization, IISWC (2009)

5. Chen, X., Chang, L.W., Rodrigues, C.I., Lv, J., Wang, Z., Hwu, W.M.: Adaptive cache management for energy-efficient GPU computing. In: Proceedings of the 47th IEEE/ACM International Symposium on Microarchitecture, MICRO (2014)
6. Hennessy, J., Patterson, D.: Computer architecture: a quantitative approach, Fifth Edition (2012)
7. Hong, S., Oguntebi, T., Olukotun, K.: Efficient parallel graph exploration on multi-core CPU and GPU. In: International conference on parallel architectures and compilation techniques, PACT (2011)
8. Huang, J.C., Lee, J.H., Kim, H., Lee, H.H.S.: GPUMech: GPU performance modeling technique based on interval analysis. In: Proceedings of the international symposium on microarchitecture, MICRO (2014)
9. Jia, W., Shaw, K.A., Martonosi, M.: Characterizing and improving the use of demand-fetched caches in GPUs. In: Proceedings of the 26th ACM international conference on supercomputing, ICS (2012)
10. Kumar, S., Wilkerson, C.: Exploiting spatial locality in data caches using spatial footprints (1998)
11. Lal, S., Lucas, J., Andersch, M., Alvarez-Mesa, M., Elhossini, A., Juurlink, B.: GPGPU Workload characteristics and performance analysis. In: Proceedings of the 14th international conference on embedded computer systems: architectures, modeling, and simulation, SAMOS (2014)
12. Lal, S., Juurlink, B.: A quantitative study of locality in GPU caches. In: Proceedings of the international conference on embedded computer systems: architectures, modeling, and simulation, SAMOS (2020)
13. Li, A., van den Braak, G.J., Kumar, A., Corporaal, H.: Adaptive and transparent cache bypassing for GPUs. In: Proceedings of the international conference for high performance computing, networking, storage and analysis, SC (2015)
14. Li, A., Song, S.L., Liu, W., Liu, X., Kumar, A., Corporaal, H.: Locality-aware CTA clustering for modern GPUs. In: Proceedings of the international conference on architectural support for programming languages and operating systems, ASPLOS (2017)
15. Li, C., Song, S.L., Dai, H., Sidelnik, A., Hari, S.K.S., Zhou, H.: Locality-driven dynamic GPU cache bypassing. In: Proceedings of the 29th ACM on international conference on supercomputing, ICS (2015)
16. Meng, J., Tarjan, D., Skadron, K.: Dynamic warp subdivision for integrated branch and memory divergence tolerance. In: Proceedings of the 37th annual international symposium on Computer architecture, ISCA (2010)
17. Narasiman, V., Shebanow, M., Lee, C.J., Miftakhutdinov, R., Mutlu, O., Patt, Y.N.: Improving GPU performance via large warps and two-level warp scheduling. In: Proceedings of the 44th annual IEEE/ACM international symposium on microarchitecture, MICRO (2011)
18. NVIDIA: CUDA: compute unified device architecture. http://developer.nvidia.com/object/gpucomputing.html (2007)
19. Reguly, I.Z., Giles, M.: Efficient sparse matrix-vector multiplication on cache-based GPUs. In: Proceedings of the innovative parallel computing, InPar (2012)
20. Rhu, M., Sullivan, M., Leng, J., Erez, M.: A locality-aware memory hierarchy for energy-efficient GPU architectures. In: Proceedings of the 46th annual IEEE/ACM international symposium on microarchitecture, MICRO (2013)
21. Rogers, T.G., O'Connor, M., Aamodt, T.M.: Cache-conscious wavefront scheduling. In: Proceedings of the 45th annual IEEE/ACM international symposium on microarchitecture, MICRO (2012)
22. Sandokji, S., Essa, F., Fadel, M.: A survey of techniques for warp scheduling in GPUs. In: Proceedings of the international conference on intelligent computing and information systems, (ICICIS) (2015)
23. Sartori, J., Kumar, R.: Branch and data herding: reducing control and memory divergence for error-tolerant GPU applications. In: Proceedings of the 21st international conference on parallel architectures and compilation techniques, PACT (2012)
24. Shi, Z., Huang, X., Jain, A., Lin, C.: Applying deep learning to the cache replacement problem. In: Proceedings of the IEEE/ACM international symposium on microarchitecture, MICRO (2019)
25. Tang, X., Pattnaik, A., Kayiran, O., Jog, A., Kandemir, M.T., Das, C.: Quantifying data locality in dynamic parallelism in GPUs. Proc. ACM Meas. Anal. Comput, Syst (2018)
26. Tarjan, D., Meng, J., Skadron, K.: Increasing memory miss tolerance for SIMD cores. In: Proceedings of the conference on high performance computing networking, storage and analysis, SC (2009)
27. Vijaykumar, N., Ebrahimi, E., Hsieh, K., Gibbons, P.B., Mutlu, O.: The locality descriptor: a holistic cross-layer abstraction to express data locality in GPUs. In: Proceedings of the international symposium on computer architecture (ISCA) (2018)

28. Wang, L., Jahre, M., Adileho, A., Eeckhout, L.: MDM: The GPU memory divergence model. In: Proceedings of the international symposium on microarchitecture (MICRO) (2020)
29. Xiao, S., Lin, H., Feng, W.C.: Accelerating protein sequence search in a heterogeneous computing system. In: IEEE International parallel and distributed processing symposium, IPDPS (2011)
30. Xie, X., Liang, Y., Wang, Y., Sun, G., Wang, T.: Coordinated static and dynamic cache bypassing for GPUs. In: 2015 IEEE 21st International symposium on high performance computer architecture, HPCA (2015)
31. Zhang, E.Z., Jiang, Y., Guo, Z., Tian, K., Shen, X.: On-the-fly elimination of dynamic irregularities for GPU computing. In: Proceedings of the international conference on architectural support for programming languages and operating systems, ASPLOS (2011)