A Model-Driven Approach to Warp/Thread-Block Level GPU Cache Bypassing

Hongwen Dai, Chao Li, Huiyang Zhou
North Carolina State University
Raleigh, NC
{hdai3,cli17,hzhou}@ncsu.edu

Saurabh Gupta, Christos Kartsaklis
Oak Ridge National Lab
Oak Ridge, TN
{guptas1, kartsaklisc}@ornl.gov

Mike Mantor
Advanced Micro Devices
Orlando, FL
Michael.Mantor@amd.com

Abstract

The high amount of memory requests from massive threads may easily cause cache contention and cache-miss-related resource congestion on GPUs. This paper proposes a simple yet effective performance model to estimate the impact of cache contention and resource congestion as a function of the number of warps/thread blocks (TBs) to bypass the cache. Then we design a hardware-based dynamic warp/thread-block level GPU cache bypassing scheme, which achieves 1.68x speedup on average on a set of memory-intensive benchmarks over the baseline. Compared to prior works, our scheme achieves 21.6% performance improvement over SWL-best [29] and 11.9% over CBWT-best [4] on average.

1. Introduction

General purpose computation on graphics processing units (GPGPU) has become prevalent in high performance computing. Modern GPUs have adopted multi-level cache hierarchies to mitigate the long off-chip memory access latency. However, the cache capacity per thread and the cache line lifetime on GPUs are much smaller than those on CPUs, resulting in significant cache trashing especially due to inter-warp contention [29]. Moreover, since miss status handling registers (MSHRs) and miss queue entries need to be allocated for outstanding misses, massive multithreading on GPUs can cause severe memory pipeline stalls when such resources are fully occupied. Simply enlarging cache capacity and/or adding more cache-miss-related resources is costly and sometimes impractical due to the required area and power.

Thread throttling [4][11][22][29][33] has been proposed based on the observation that intra-warp locality is crucial for highly cache sensitive workloads. Cache Conscious Wavefront Scheduling (CCWS) [29] improves performance by limiting the number of actively scheduled warps, thereby reducing L1 D-cache thrashing and preserving intra-warp locality.

However, latency hiding via massive multithreading is affected by warp throttling due to the reduced number of active warps. Furthermore, warp throttling may cause Network-on-Chip (NoC) and DRAM bandwidth to be underutilized. Cache bypassing [8][19][31] has been applied to protect 'hot' cache lines from early eviction. However, with so many threads sharing caches on GPUs, it is difficult to make robust predictions on hot cache lines. Coordinated Bypassing and Warp Throttling (CBWT) [4] combines protection distance based bypassing (PDP) [7] and warp throttling to overcome the limitations of cache-bypassing-only or warp-throttling-only solutions.

In this paper, we propose a simple yet effective performance model to estimate the number of cache hits and reservation failures due to cache-miss-related resource congestion as a function of the number of warps/thread-blocks to access/bypass the cache. Based on the model, we design a cost-effective hardware-based dynamic scheme to find the optimal number of warps/TBs to bypass the L1 D-cache. The key difference from prior works on GPU cache bypassing is that we do not rely on accurate prediction on hot cache lines. Compared to warp throttling, we do not limit the number of active warps so as to exploit the available thread-level parallelism (TLP) and otherwise underutilized NoC and off-chip memory bandwidth. Furthermore, our scheme is simple to implement and does not alter the existing cache organization. Overall, this paper makes the following contributions:

(1) We characterize cache behaviours of GPGPU applications and show that cache contention and cache-miss-related resource congestion should be considered jointly for GPU cache bypassing.

(2) We propose a simple yet effective performance model to estimate the number of cache hits and reservation failures to find the optimal number of warps/TBs to bypass the L1 D-cache. Based on the model, we design a dynamic GPU cache bypassing scheme.

(3) Our scheme achieves significant performance improvements over the baseline and outperforms the state-of-the-art GPU cache management techniques, including CCWS and CBWT.

DAC '16, June 05-09, 2016, Austin, TX, USA
© 2016 ACM. ISBN 978-1-4503-4236-0/16/06, $15.00
DOI: http://dx.doi.org/10.1145/2897937.2897966

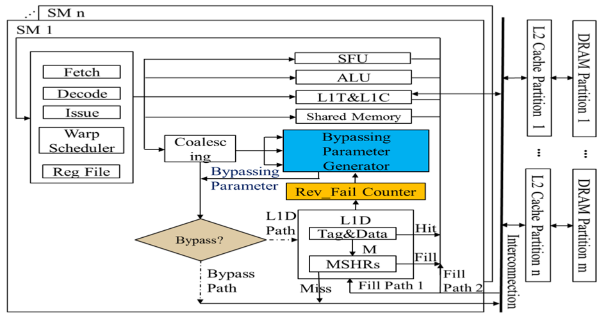


Figure 1. Baseline GPU. Our proposed dynamic scheme adds a Bypassing Parameter Generator and minor change in the bypassing decision logic.

2. Background and Related Work

2.1 Baseline Architecture

A GPU kernel is launched with a grid of thread blocks (TBs). Threads within a TB form multiple warps and all threads in a warp execute in a lock step manner. One GPU consists of multiple Streaming Multiprocessors (SMs). As shown in Figure 1, on each SM, there are a L1 read-only texture cache and a constant cache, a data cache (L1 D-cache) and shared memory. A unified L2 cache is shared among multiple SMs. Typically, the L1 D-cache uses write-through with either write-allocate [1] or write-no-allocate [24][26] policies, and the L2 cache uses the write-back write-allocate to save the DRAM bandwidth [30].

2.2 Baseline Memory Request Handling

On GPUs, global and local memory requests from threads in a warp are coalesced into as few transactions as possible before being sent to the memory hierarchy. The cached or bypassed information is typically encoded in instruction opcodes [27], indicating whether a

Table 1. Baseline architecture configuration

# of SMs	15, SIMD width=32, 1.4GHz
Per-SM warp schedulers	2 round-robin warp schedulers
Per-SM limit	1536 threads, 48 warps, 8 thread blocks, 32 MSHRs, 8 miss queue entries
Per-SM L1D-cache	16KB, 128B line, 4-way associativity
Per-SM shared	48KB, 32 banks
Unified L2 cache	768 KB, 128KB/partition, 6 partitions, 128B line, 16-way associativity
L1D/L2 policies	alloc-on-miss, LRU, L1D:WTWN, L2: WBWA
Interconnect	32B channel width, 1.4GHz
DRAM	6 memory channels, FR-FCFS scheduler, 924MHz, BW: 48bytes/cycle

Table 2. Benchmarks

Name	Description	Type	Suite
PTF	ParticleFilter	HCC	[5]
SC	StreamCluster	HCC	[5]
BH	Barnes-Hut	HCC	[3]
HST	Histogram	HCC	[20]
S2K	Symmetric rank-2k operations	HCC	[10]
KMS	K-means clustering	HCC	[5]
GMV	Scalar-vector-matrix multiply	HCC	[10]
ATX	Matrix-transpose-vector	HCC	[10]
MVT	Matrix-vector-product	HCC	[10]
BIC	BiCGStab linear solver	HCC	[10]
SCP	ScalarProduct	LCC	[20]
CVR	Covariance Computation	LCC	[10]
LUD	LU Decomposition	LCC	[10]
SR2	Srad2	LCC	[10]
2DF	2D Finite different time domain	LCC	[10]
2DV	2D Convolution	LCC	[10]

request is sent to the L1 D-cache through the 'L1D path' or directly sent to the L2 cache through the 'Bypass Path', as shown in Figure 1. For a request sent to the L1 D-cache, if it is a miss, the resources including a cache line slot, a MSHR entry and a miss queue entry need to be allocated. If any of these cache-miss-related resources is not available, a reservation failure occurs and the memory pipeline is stalled. The MSHR entry is reserved until the data is fetched from the L2 cache/off-chip memory while the miss queue entry is released when the miss request is forwarded to the L2 cache. Compared to bypassed requests, a request send to the L1 D-cache enjoys low access latency if it hits in the L1 D-cache. However, massive requests sent to the L1 D-cache can easily cause cache thrashing and cache-miss-related resource congestion, thereby degrading the overall performance.

In our warp/TB level cache bypassing scheme, once a warp/TB is determined to bypass the L1 D-cache, all its accesses go to the 'Bypass Path'. This is straightforward to implement, compared to the schemes predicting and bypassing zero/low reused cache lines.

2.3 Related Work

Guz et al. [11] demonstrate that increasing the number of threads accessing a cache can improve performance until the aggregate working set no longer fits in the cache. Kayiran et al. [16] and Xie et al. [33] dynamically adjust the number of TBs accessing L1 D-caches. Rogers et al. [29] propose CCWS to control the number of actively scheduled warps. In comparison, our scheme adaptively works on either the warp or TB level and find the optimal number of warps/TBs to access the L1 D-cache. The rest warps/TBs are not suspended. Instead they simply bypass the L1 D-cache to leverage the otherwise underutilized NoC and memory bandwidth. In other words, we do not penalize TLP like thread throttling approaches.

On GPU cache management, Jia et al. propose MPRB [15] to preserve intra-warp locality and bypasses the cache when there are memory pipeline stalls. Detecting and protecting hot cache lines has also been proposed [19][31]. However, irregular memory access patterns make accurate detection challenging.

Recent works have also exploited the combination of thread throttling and cache bypassing. Li et al. [22] propose priority-based cache allocation on top of CCWS. Chen et al. propose CBWT [4] that adopts PDP for L1 D-cache bypassing and applies warp throttling if the contention on the L2 cache and DRAM is severe. However, PDP is not as effective as it is on CPUs. Li et al [18] propose a compile-time framework for cache bypassing at the warp level for global memory reads.

The aforementioned works either require multiple profiling runs [4][18][22][29] or incur non-trivial hardware overhead due to reorder queues [15], warp throttling detection [4][29][33], finite state machines [22], and cache line protection schemes [4][19][31]. In contrast, our sampling-based scheme is much simpler to implement and the results in Section 7 show that it can achieve higher performance than the state-of-the-art techniques.

Several GPU performance models have been proposed. Hong et al. [13] use memory warp parallelism and computation warp parallelism to estimate the performance. Zhang et al. [34] develop a micro-benchmark based performance model to measure the execution time of the instruction pipeline, shared memory accesses, and global memory accesses. Huang et al. [14] propose GPUMech, which profiles the instruction trace of every warp and models multithreading and resource contentions caused by memory divergence. In comparison, our model captures cache contention and cache-miss-related resource congestion as a function of the number of warps/TBs accessing the cache, which has not been addressed in prior models.

3. Experimental Methodology

Simulation Environment: We use GPGPUsim V3.2.1 [2], a cycle-accurate GPU microarchitecture simulator, to evaluate our proposed scheme. The baseline GPU architecture configuration is shown in Table 1, based on the Fermi architecture. To reduce the conflict misses, we experimented different cache indexing functions [9][17] and choose to use the bitwise-XOR indexing function as baseline for its simplicity and effectiveness.

Benchmarks: We evaluate a collection of benchmarks, listed in Table 2, from Rodinia [5], Polybench [10] and LonestarGPU [3], including both regular and irregular applications. HST and SCP are the data cache version adopted from [20]. We evaluate all these workloads with their default grid/block dimensions/inputs, scaled if necessary, similar to [15][23]. All simulations run to completion.

4. Characterization and Motivation

We first analyse the performance impact of the L1 D-cache. Figure 2 shows the performance under different L1 D-cache configurations, including: (1) a 16KB L1 D-cache with default modulo cache indexing, (2) a 16KB L1 D-cache with bitwise-XOR cache indexing, (3) no L1 D-cache, and (4) an 8MB L1 D-cache.

We make the following observations from Figure 2. First, it demonstrates that the indexing function has a remarkable performance impact. On average, the bitwise-XOR indexing function has almost 2X speedup over the modulo indexing function. Second, the 16kB L1 D-cache improves the performance for the benchmarks PTF, HST, SR2, 2DF and 2DV. On the other hand, many other benchmarks, including SC, S2K, KMS etc., show better

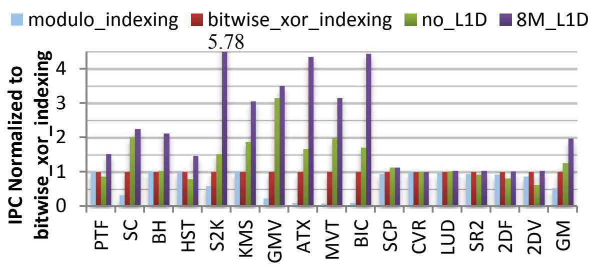


Figure 2. Performance under different L1 D-cache configurations.

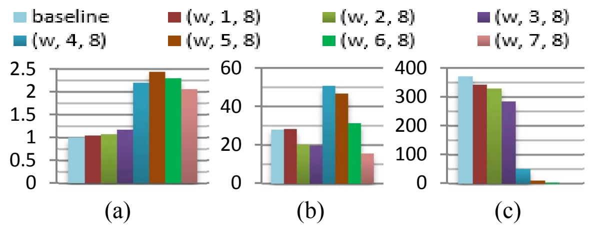


Figure 3. Bypassing various numbers of warps for KMS: (a) Normalized IPC, (b) L1 D-cache hits per Kilo-Inst and (c) Reservation Failures per Kilo-Inst (RFKI)

performance when there is no L1 D-cache. The reason is that they do not suffer from cache-miss-related resource congestion and the resulting memory pipeline stalls. Third, an 8MB L1 D-cache greatly improves the performance by reducing both cache contention and resource congestion.

Based on the performance improvement obtained from an 8MB L1 D-cache, we classify the benchmarks into two categories: High Cache Contention (HCC) ones with more than 50% performance improvement over the baseline and Low Cache Contention (LCC) ones with less than 50% improvement (Figure 2 and Table 2). Although we focus on HCC benchmarks, the results of LCC ones are included to show the robustness of our scheme.

Next, we look into warp/TB level GPU cache bypassing as the intra-warp locality is preserved. We use **(w, M, N)** to denote bypassing *M* of *N* warps; and **(b, M, N)** for bypassing *M* of *N* TBs. An interesting case study on the benchmark KMS is shown in Figure 3. Figure 3(a) shows the performance initially increases and then decreases with more warps being bypassed. The reason is that although the L1 D-cache contention is further relieved with more warps bypassed, the cache utilization is affected. As shown in Figure 3(b), (w, 4, 8) has more hits than (w, 5, 8). On the other hand, (w, 4, 8) incurs more reservation failure than (w, 5, 8) as shown in Figure 3(c). The combined effect leads to the fact that (w, 5, 8) achieves the best performance. From this case study, we conclude that a metric integrating both cache hits and reservation failures is needed to determine the optimal number of warps/TBs to access/bypass the cache.

5. A Performance Model Integrating Cache Contention and Related Resource Congestion

In this section, we present our model to determine how many warps/TBs to bypass the L1 D-cache. First, we use the number of cache hits instead of hit rates to model cache contention. Then, we propose an empirical model to estimate the impact of cache-miss-related resource congestion. Next, we define a new '*Adjusted Hits*' metric integrating both cache hits and reservation failures. The optimal number of warps/TBs to bypass the L1 D-cache is the one maximizing this new metric.

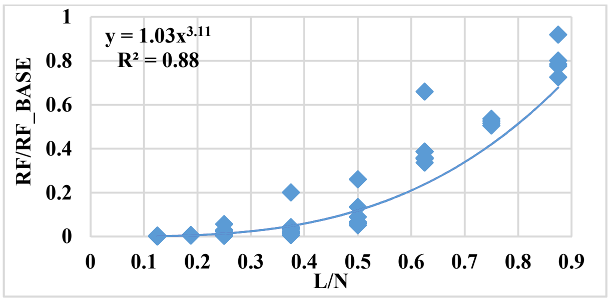


Figure 4. Identifying the relationship between the (normalized) reservation fails and the (normalized) number of warps/TBs accessing the L1 D-cache.

5.1 Modelling Cache Contention

Although the cache hit rate shows how often the cached data is reused, it does not correlate well with the overall performance because it is oblivious to the number of accesses, which may vary due to bypassing. Therefore, we use the number of hits to measure the performance impact of cache contention.

5.2 Modelling Cache-Miss-Related Resource Congestion

To model cache-miss-related resource congestion, we propose an empirical model capturing the relationship between the number of reservation failures and the number of bypassed warps/TBs.

Our empirical model is constructed using curve fitting, similar to the studies on cache miss rates vs. cache capacity [6][12]. To this end, we run simulations with various numbers of warps/TBs bypassing the L1 D-cache for the 5 HCC benchmarks from Polybench [10], and collect the numbers of reservation failures. Specifically, if there are more than one TB running on an SM, we vary the number of TBs to bypass the L1 D-cache; if there is only one TB on an SM, we vary the number of warps. Assume the total number of reservation failures on an SM is *RF_base* for the baseline, i.e. when all N warps/TBs access the L1 D-cache. The goal of our model is to calculate *RF_estimated*, the estimated number of reservation failures when *M* warps/TBs bypass the L1 D-cache, i.e., when *L* warps/TBs access the L1 D-cache, where *L*= *N-M*. Figure 4 shows that the number of reservation failures (normalized to *RF_base*) varies with the number of warps/TBs (normalized to *N*) accessing the L1 D-cache. Then, we perform curve fitting and identify a cubic relationship between (*RF_estimated*/*RF_base*) and (*L*/*N*). The coefficient of determination, R^2, is 0.88, indicating the formula is indicative enough to use as our goal is to match the trend rather than reproducing the exact number. Based on the cubic relationship, we estimate the number of reservation failures as following:

$$RF\_estimated = RF\_base * (L/N)^3$$

5.3 Putting It All Together

The goal of optimal bypassing is to maximize cache hits while minimizing cache-miss-related resource congestion. To achieve this, we propose a new metric '*Adjusted Hits*' to incorporate both the number of hits and reservation failures:

$$Adjusted\ Hits = Hits\_estimated - RF\_estimated*E$$

Here *Hits_estimated* is obtained from sampling in our dynamic bypassing scheme. As described in Section 5.2, we profiled 5 HCC benchmarks from Polybench and found that when E is in the range of 0.3 to 0.7, the metric '*Adjusted Hits*' correlates well with the overall performance (i.e. IPC). Therefore, we choose E as 0.5 as it can be implemented with a shift operation. The optimal bypassing parameter is the one maximizing '*Adjusted Hits*'.

6. Dynamic Warp/TB Level Bypassing

Based on the empirical model presented in Section 5, we propose a bypassing parameter generator (BPG). Figure 5 shows the organization of a BPG, which has a set of '*Adjusted Hits*' calculators and a bypassing parameter selector.

An '*Adjusted Hits*' calculator estimates the number of hits and reservation failures. It uses an online monitor to compute the number of hits. The online monitor is essentially a shadow tag array, which has the same structure as the tag store in the L1 D-cache. We adopt set sampling [28] to mitigate the hardware cost. In our implementation, 4 out of 32 sets (i.e. sampling ratio 1:8) are sampled. An '*Adjusted Hits*' calculator obtains the number of reservation failures from the single global reservation failure counter as shown in Figure 1 and then uses the model described in Section 5.2 to calculate *RF_estimated* when bypassing a specific number of warps/TBs. This module is the Rev_Failure Estimator shown in Figure 5.

Each '*Adjusted Hits*' calculator is dedicated to tracking a specific number of warps/TBs. For instance, the first '*Adjusted Hits*' calculator generates the '*Adjusted Hits*' for warp/TB 0, the second one works for warps/TBs 0~1 and so on. In a BPG, there is a set of 8 '*Adjusted Hits*' calculators supporting 1 to 8 warps/TBs. The reason is that there can be up to 8 TBs on one SM on the Fermi architecture and generally no more than 8 warps (256 threads) within a TB. Although 8 '*Adjusted Hits*' calculators seem to be limiting when there are more than 8 warps in a TB (e.g. PTF has 16 warps in one TB), our experiments show the optimal number of warps/TBs accessing L1 D-cache is smaller than 8 when there is opportunity for bypassing. So we keep the hardware overhead small by limiting the number of calculators to 8 in a BPG.

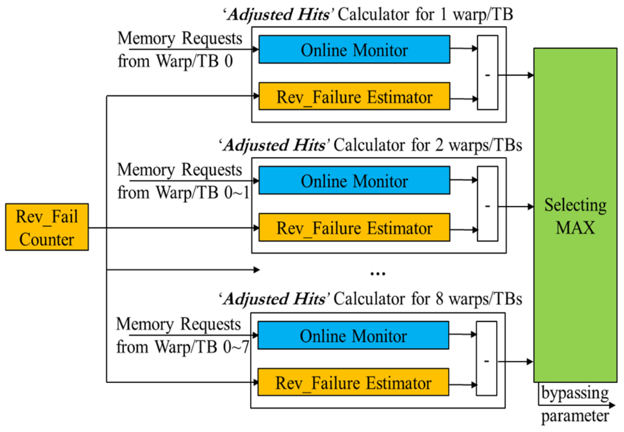
Considering the complexity of the cubic operation and the small range of bypassing parameters (8 in total), each Rev_Failure Estimator is implemented as a 1-entry pre-computed lookup table for $(L/N)^3$ and a multiplier for $RF\_base * (L/N)^3$.

After the '*Adjusted Hits*' calculators produce '*Adjusted Hits*' for different numbers of warps/TBs accessing the L1 D-cache, we select the one, L_{max}, that results in the highest '*Adjusted Hits*', and have the first L_{max} warps/TBs access L1 D-cache while bypassing the rest. If the BPG figures out that it is best to have 8 warps/TBs access L1 D-cache, it means no opportunity for warp/TB bypassing and all warps/TBs continue to access L1 D-cache.

In our experiment, a new bypassing parameter is produced every 1000 accesses from all warps/TBs on one SM. The sampling interval of 1000 accesses works well in capturing the phase behaviour. In order to not completely lose the history information, the hit counters and the reservation failure counter are right shifted by one bit every time a new bypassing parameter is created.

We use CACTI 6.5 [32] to evaluate the hardware cost of BPG. The majority source of area overhead is the shadow tag arrays used in '*Adjusted Hits*' calculators. Each shadow tag array entry is 35-bits (1 valid bit + 2 LRU bits + up to 32 bits tag). With 8 '*Adjusted Hits*' calculators on each SM, the total area for shadow tag arrays is estimated as 0.087mm^2 for 15 SMs using the 45nm technology. Such additional area is approximately 0.016% of the GTX480 area [26], which is a 15-SM system with the 40nm technology. Other hardware costs include one 10-bit hit number counter for each '*Adjusted Hits*' calculator, one global 12-bit reservation failure counter on each SM, etc. Those miscellaneous overheads are negligible in comparison. Since there is one BPG per SM, we refer to this design as local BPG or L-BPG.

In the L-BPG design, the hardware overhead is proportional to the number of SMs. To reduce the hardware cost, we propose a global BPG design, G-BPG, which monitors application behaviors on one SM and applies the bypassing parameter to other SMs. The bypassing parameter broadcasting is not on the critical path and can be done with simple logic added to the existing TB-dispatcher. As shown in Figure 7.1, G-BPG is able to reap most performance improvement of L-BPG.



'*Adjusted Hits*' Calculator for 1 warp/TB
Memory Requests from Warp/TB 0
Online Monitor
Rev_Failure Estimator
-
'*Adjusted Hits*' Calculator for 2 warps/TBs
Memory Requests from Warp/TB 0~1
Rev_Fail Counter
...
'*Adjusted Hits*' Calculator for 8 warps/TBs
Memory Requests from Warp/TB 0~7
Selecting MAX
bypassing parameter

Figure 5. The organization of a BPG.

We also use CACTI 6.5 [32] to evaluate the energy efficiency of the shadow tag arrays. The dynamic read energy per access and total leakage power is estimated as 0.001 nJ and 0.27 mW, respectively. Such small energy overhead is negligible compared to the substantial static energy savings from the significantly reduced execution time.

7. Experimental Results and Analysis

In this section, we conduct experimental analysis on our bypassing scheme and compare them with state-of-the-art techniques.

7.1 Performance Evaluation and Analysis

In our evaluation, we compare our Model-Driven Bypassing (MDB) scheme to two closely related techniques. All performance results are normalized to the baseline, which sends all memory requests to L1 D-cache (i.e. no bypassing). The bitwise-XOR mapping function is used for both L1 D-caches and the L2 cache.

SWL-best: The optimal Static Wavefront Limiting (SWL) configuration. As demonstrated in the prior work [29], SWL-best outperforms dynamic CCWS due to the start-up cost associated with the latter.

CBWT-best: The optimal SWL with PDP bypassing enabled. As demonstrated by Chen et al. [4], CBWT outperforms pure bypassing/warp-throttling and CBWT-best performs better than dynamic CBWT.

MDB-local: Our proposed dynamic warp/TB level GPU bypassing scheme, which deploys one BPG per SM.

MDB-global: Our proposed dynamic warp/TB level GPU bypassing scheme, which adopts the global BPG design.

a) Performance Comparison.

Figure 6 shows performance of different GPU cache management schemes. First, it shows the effectiveness of MDB. Although the model (Section 5.2) is derived from curve fitting among a subset of HCC benchmarks, both MDB-local and MDB-global improve the performance of all HCC benchmarks and show no performance degradation for the LCC benchmarks. We also confirm that the cubic relationship, identified in Section 5.2, holds in general for all HCC benchmarks. GM_HCC represents the average (geometric mean) performance of HCC benchmarks in Figure 6.

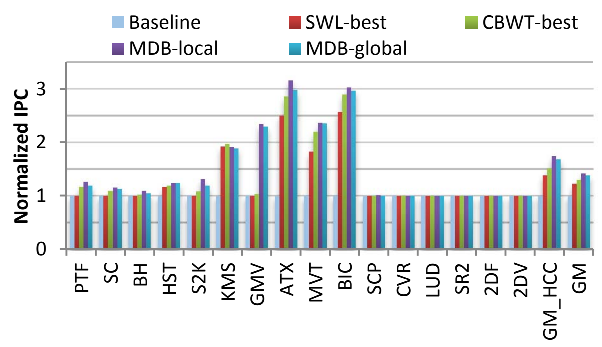


Figure 6. Performance of different GPU cache management schemes.

Comparing MDB-global with MDB-local, we observe that MDB-global can reap most of the benefits of the latter with significantly less hardware overhead. MDB-global achieves an average of 1.69x speedup over the baseline, close to the 1.75x speedup from MDB-local. It shows a mild performance loss compared to MDB-local due to the diverse runtime behaviors on different SMs, for the benchmarks such as PTF, S2K, etc. In the following discussion, we focus on MDB-global.

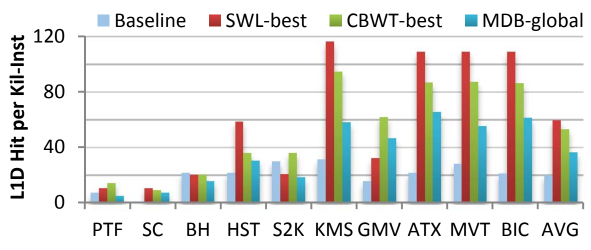
Figure 6 also demonstrates that our MDB approach outperforms the prior GPU cache management schemes. While SWL-best and CBWT-best achieve an average of 1.39x and 1.51x speedup over the baseline, respectively, MDB-global achieves an average of performance improvement of 21.6% over SWL-best and 11.9% over CBWT-best. Next, we dissect the reasons. We focus on HCC benchmarks as all these schemes have small impact on LCC ones.

b) Hits & Reservation Failures per Kilo-Instruction

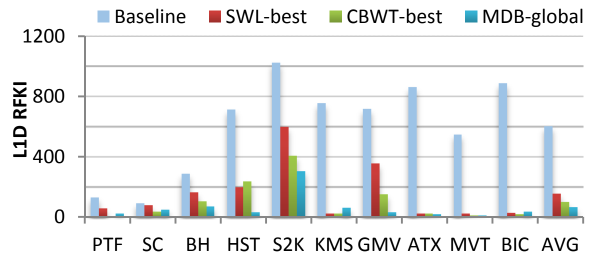
As discussed in Section 4, both cache hits and reservation failures due to cache-miss-related resource congestion affect the overall performance. Here, we report the number of cache hits and reservation failures in Figure 7 (a) and (b). We make several observations from the figure. First, SWL-best significantly increases the number of hits and reduces the number of reservation failures for HST, KMS, ATX, MVT and BIC while having minor impact on other benchmarks, leading to higher performance. Second, CBWT-best is more effective than SWL-best by combining warp throttling and the PDP cache bypassing scheme. CBWT-best improves the performance of all HCC benchmarks. For those benchmarks where SWL-best has minor impact, CBWT-best either brings more hits, e.g., for PTF, S2K and GMV, or has fewer reservation failures, e.g. for SC. MDB-global approach explicitly models reservation failures, thereby being more effective than CBWT-best. On average, MDB-global has the lowest reservation failures. MDB-global experiences more reservation failures than CBWT-best on KMS because there is a start-up delay in MDB-global and KMS is sensitive to the timing of bypassing.

c) MIPC, L1-L2 Traffic, and L2-DRAM Traffic

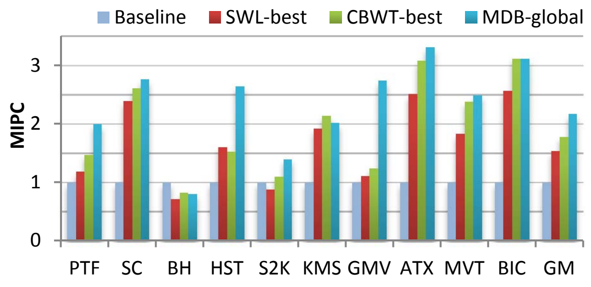
Since both the "L1D Path" and "Bypass Path" can be used to serve data, we use MIPC (Memory Instructions served Per Cycle) to capture the complete picture of memory subsystem performance, as shown in Figure 7(c). First, we can see that the trend of MIPC matches the overall performance. Second, although SWL-best has high hits and low reservation failures per kilo-instruction, it performs worse than CBWT-best and MDB because it sacrifices TLP and MLP (memory-level parallelism). In addition to warp throttling, CBWT-best adopts PDP to relieve cache contention and leverages the underutilized NoC and DRAM bandwidth. In this way, CBWT-best has a higher MIPC than SWL-best. Compared to



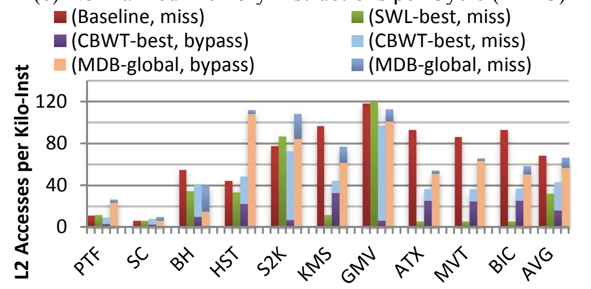
(a) L1 D-cache hits per Kilo-Inst



(b) L1 D-cache Reservation Failures per Kilo-Inst (RFKI)



(c) Normalized Memory Instructions per Cycle (MIPC)

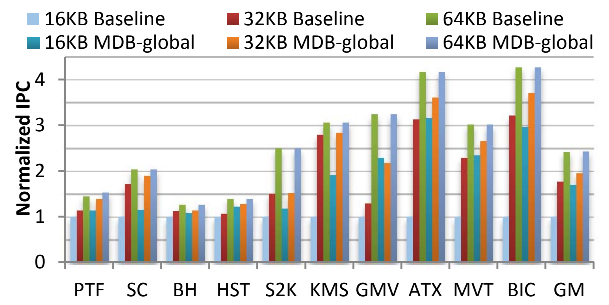


(d) L2 accesses per Kilo-Inst, sourced from L1 D-cache bypassed accesses and misses

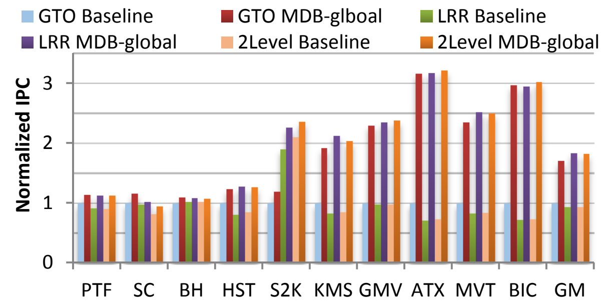
Figure 7. L1 D-cache performance and L1-L2 traffic.

SWL-best and CBWT-best, our MDB scheme does not necessarily have higher hits per kilo-instructions, but it shows lower RFKI and higher MIPC on average due to aggressive bypassing.

Figure 7(d) shows the L1-L2 traffic, sourced from L1 D-cache misses and bypassed accesses. SWL-best reduces the L1-L2 traffic from the improved L1 D-cache efficiency for KMS, ATX, etc. Compared to SWL-best, CWBT-best shows heavier L1-L2 traffic on KMS, ATX and so on, lighter on S2K and GMV, and similar on PTF, depending on the combined effect of higher L1 D-cache efficiency and cache bypassing. MDB-global reduces L2 traffic for KMS, GMV and so on but shows heavier L2 traffic for PTF, HST and S2K. The majority of L2 traffic of MDB-global is from bypassed accesses. The fact that MDB-global is aggressive on cache bypassing leads to its effectiveness in preserving intra-warp locality, improving cache efficiency, reducing cache-miss-related resource congestion, and achieving higher performance. In



(a) Performance with different L1 D-cache capacities, normalized to a 16KB L1 D-cache.



(b) Performance under different warp scheduling policies, normalized to GTO.

Figure 8. Sensitivity study.

addition, SWL-best, CBWT-best and our MDB approach all achieve lighter DRAM traffic, compared to the baseline.

7.2 Sensitivity Study

a) Sensitivity to L1 D-cache Capacity

Figure 8(a) shows the sensitivity of MDB-global on different L1 D-cache capacities. MDB-global shows performance improvement for each cache capacity, even the large ones. On the other hand, with a larger cache capacity, the performance improvement is reduced. Moreover, we can see a 16KB L1 D-cache with our proposed MDB-global can achieve similar performance to a 32KB L1 D-cache. This highlights the cost/area effectiveness of our approach, compared to enlarging the L1 D-cache capacity.

b) Sensitivity to Warp Scheduling Policies

All the experiments discussed so far use the Greedy-Then-Oldest (GTO) warp scheduling policy, which runs a single warp until a long latency operation and then picks up the oldest one. GTO improves performance by preserving intra-warp locality and performs better than Two-Level warp scheduling [23] and Round-Robin (RR) policies. Nevertheless, Figure 8(b) shows that MDB-global continuously improves the performance despite the variations when different warp scheduling policies are applied.

8. Conclusion

Throughput-oriented GPGPUs hide long operation latency with massive multithreading. However, limited per thread cache capacity and massive memory requests can easily cause cache thrashing and memory pipeline stalls. In this paper, we propose a performance model for L1 D-cache contention and cache-miss-related resource congestion. Based on the model, we design a cost-effective dynamic warp/TB level GPU cache bypassing scheme. The experimental results show that our scheme achieves significant performance improvement over the baseline and outperforms the state-of-the-art GPU cache management schemes. We also demonstrate that our scheme remains effective with different cache capacities and various warp scheduling policies.

References

[1] AMD GCN Architecture White paper, 2012.
[2] A. Bakhoda et al. Analyzing CUDA workloads using a detailed GPU simulator. In *Proceedings of ISPASS*, 2009.
[3] M. Burtscher et al. A quantitative study of irregular programs on GPUs. In *Proceedings of IISWC*, 2012.
[4] X. Chen et al. Adaptive cache management for energy-efficient GPU computing. In *Proceedings of MICRO*, 2014.
[5] S. Che et al. Rodinia: A benchmark suite for heterogeneous computing. *In Proceedings of IISWC*, 2009.
[6] C. K Chow. Determination of cache's capacity and its matching storage hierarchy. Computers, IEEE Transactions on 100, 1976.
[7] N. Duong et al. Improving cache management policies using dynamic reuse distances. In *Proceedings of MICRO*, 2012.
[8] J. Gaur et al. Bypass and insertion algorithms for exclusive last-level caches. In *Proceedings of ISCA*, 2011.
[9] A. González et al. Eliminating cache conflict misses through XOR-based placement functions. In *Proceedings of ICS*, 1997.
[10] S. Grauer-Gray et al. Auto-tuning a high-level language targeted to GPU codes. In *Proceedings of InPar*, 2012.
[11] Z. Guz et al. Many-core vs. many-thread machines: Stay away from the valley. *Computer Architecture Letters 8*, no. 1, 2009.
[12] A. Hartstein, et al. Cache miss behavior: is it√ 2?." *Proceedings of the 3rd conference on Computing frontiers*, 2006.
[13] S. Hong et al. An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness. In *Proceedings of ISCA*, 2009.
[14] J. Huang et al. GPUMech: GPU Performance Modeling Technique Based on Interval Analysis. In *Proceedings of MICRO*, 2014.
[15] W. Jia et al. MRPB: Memory request prioritization for massively parallel processors. In *Proceedings of HPCA*, 2014.
[16] O. Kayıran et al. Neither more nor less: optimizing thread-level parallelism for GPGPUs. In *Proceedings of PACT*, 2013.
[17] M. Kharbutli et al. Using prime numbers for cache indexing to eliminate conflict misses. In *Software, IEE Proceedings-*, 2004.
[18] A. Li et al. Adaptive and transparent cache bypassing for GPUs. In *Proceedings of SC*, 2015.
[19] C. Li et al. Locality-Driven Dynamic GPU Cache Bypassing. In *Proceedings of ICS*, 2015.
[20] C. Li et al. Understanding the tradeoffs between software-managed vs. hardware-managed caches in GPUs. In *Proceedings of ISPASS*, 2014.
[21] J. Leng et al. GPUWattch: enabling energy optimizations in GPGPUs. In *Proceedings of ISCA*, 2013.
[22] D. Li et al. Priority-based cache allocation in throughput processors. In *Proceedings of HPCA*, 2015.
[23] V. Narasiman et al. Improving GPU performance via large warps and two-level warp scheduling. In *Proceedings of* MICRO, 2011.
[24] NVIDIA Kepler GK110 white paper. 2012.
[25] NVIDIA, "CUDA C/C++ SDK code samples,"2011.
[26] NVIDIA's CUDA compute architecture: Fermi. 2009.
[27] NVIDIA Parallel Thread Execution ISA Version 4.2.
[28] M. Qureshi et al. Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches." In *Proceedings of* MICRO, 2006.
[29] T. Rogers et al. Cache-conscious wavefront scheduling." In *Proceedings* MICRO, 2012.
[30] I. Singh et al. Cache coherence for GPU architectures. In *Proceedings of HPCA*, 2013.
[31] Y. Tian et al. Adaptive GPU cache bypassing. In Proceedings of the 8th Workshop on General Purpose Processing using GPUs, 2015.
[32] S. Wilton et al. "CACTI: An enhanced cache access and cycle time model." Solid-State Circuits, IEEE Journal of 31, no. 5 (1996).
[33] X. Xie et al. Coordinated static and dynamic cache bypassing for GPUs. In *Proceedings of HPCA*, 2015.
[34] Y. Zhang et al. A quantitative performance analysis model for GPU architectures. In *Proceedings of HPCA,* 2011.