2016 IEEE 22nd International Conference on Parallel and Distributed Systems

Selectively GPU Cache Bypassing for Un-Coalesced Loads

Chen Zhao[1], Fei Wang[1], Zhen Lin[2], Huiyang Zhou[2], Nanning Zheng[1]
[1]Institute of Artificial Intelligence and Robotics, Xi'an JiaoTong University, Xi'an, China
[2]Department of Electrical and Computer Engineering, North Carolina State University, Raleigh, USA
{chenzhao, wfx, nnzheng}@mail.xjtu.edu.cn
{zlin4, hzhou}@ncsu.edu

***Abstract*—GPUs are widely used to accelerate general purpose applications, and could hide memory latency through massive multithreading. But multithreading can increase contention for the L1 data caches (L1D). This problem is exacerbated when an application contains irregular memory references which would lead to un-coalesced memory accesses. In this paper, we propose a simple yet effective GPU cache Bypassing scheme for Un-Coalesced Loads (BUCL). BUCL makes bypassing decisions at two granularities. At the instruction-level, when the number of memory accesses generated by a non-coalesced load instruction is bigger than a threshold, referred as the threshold of un-coalescing degree (TUCD), all the accesses generated from this load will bypass L1D. The reason is that the cache data filled by un-coalesced loads typically have low probabilities to be reused. At the level of each individual memory access, when the L1D is stalled, the accessed data is likely with low locality, and the utilization of the target memory sub-partition is not high, this memory access may also bypass L1D. Our experiments show that BUCL achieves 36% and 5% performance improvement over the baseline GPU for memory un-coalesced and memory coherent benchmarks, respectively, and also significantly outperforms prior GPU cache bypassing and warp throttling schemes.**

***Keywords*—GPU; Cache Bypassing; Memory divergence; Un-Coalesced Load Instruction; Data Cache**

I. Introduction

Graphics Processing Units (GPUs) as throughput-oriented processors are increasingly used for general-purpose computation in recent years. GPUs leverage massively thread-level parallelism (TLP) to hide long memory access latency, and could offer dramatic performance improvement and better energy efficiency than multicore CPUs for programs mapped well to GPU hardware. CUDA [1] and OpenCL [2] make the general-purpose computation on GPU (GPGPU) pervasive in many disciplines, including image processing, pattern recognition, neural networks, etc.

Data caches especially L1 data (L1D) caches in GPUs are important for reducing memory access latency and meeting the high bandwidth requirements. But because of massive multithreading in GPUs, the working set of an application often exceeds the L1D capacity. For instance, 1536 threads (48 warps) and 2048 threads (64 warps) are supported in each streaming multiprocessor (SM) in the Fermi and Kepler architecture, respectively. However, the capacity of L1D shared by these threads is only 16KB or 48KB [3, 4]. To reduce the number of memory accesses, the memory coalescing unit is integrated into the load-store unit of an SM such that when a memory access instruction is executed, the individual memory requests from the multiple (32) threads in a warp can be combined into few memory transactions. However, memory coalescing is only effective when the memory references of the threads in a warp are regular and could be coalesced into few cache blocks. For GPGPU applications with irregular memory references, such as graph-based and data-mining applications, the accesses of the threads in a warp often could not be coalesced into one or two cache blocks. Such un-coalesced memory accesses often lead to memory divergence [5, 6], which means that for one memory access instruction, some threads of a warp observe low latency due to cache hits while other threads experience longer latencies due to cache misses. Due to the SIMD-style execution, all the threads in the warp are forced to wait for the slowest memory access to finish. The problem due to un-coalesced memory accesses are analyzed and discussed in [7, 8]. Un-coalesced load or store instructions also result in massive memory accesses, which aggravate the L1D contention and cause frequent cache thrashing and serious cache pollution. In this case, L1D often becomes the performance bottleneck. Furthermore, the prior cache management and replacement schemes for CPUs such as RRIP [9] and DIP [10], are not effective on GPUs [11], since they are not designed for GPU massive multithreading.

Cache bypassing is proven to be an effective way to alleviate L1D contention for GPUs in recent studies [11-18]. It allows memory requests to access lower levels of memory hierarchy rather than waiting previous ones to finish accessing L1D. Compiler-based cache bypassing is proposed in [16, 17]. Adaptive cache bypassing schemes according to dynamic locality estimation are introduced in [11, 15, 18], but these methods need extra on-chip memory tables to track the memory access pattern. In MRPB [14], memory accesses bypass L1D when L1D is stalled. The drawback is that data locality is not considered. Another way to make bypass decisions is to enhance the L2 cache to detect the contention and feed the information back to L1D [12, 13]. Although un-coalesced accesses are an important reason for L1D contention and thrashing, none of above bypassing schemes directly targets at this root cause. An L1D management policy according to the warp's scheduling priority and memory divergence is proposed in [19], which also needs an extra victim cache to track inter-warp locality. Cache bypassing and insertion policies based on inter-warp heterogeneity are introduced in [20]. It mainly focuses on the L2 cache while it is L1D that usually suffers more severe contention for most GPGPU applications.

In this paper, a simple yet effective bypassing scheme targeting at un-coalesced loads is proposed. At the instruction level, all memory accesses of an un-coalesced load would bypass L1D when the number of the accesses generated from this un-coalesced load for the threads in a warp is larger than a pre-determined threshold, TUCD. At the level of individual memory accesses, one memory access could also bypass L1D when L1D is stalled; the fetched data is likely to have poor locality; and the utilization on destination memory sub-partition is not high. Overall, this paper makes the following contributions:

(1) We analyze the L1D performance for load instructions with different memory un-coalescing degrees in both memory-coherent (i.e., memory coalesced) and memory un-coalesced workloads.
(2) We propose a two-granularity bypassing scheme referred as BUCL, such that the scare L1D resources are preserved for the data with relatively high locality.
(3) We show that BUCL achieves significant performance improvement over the GPU baseline architecture, meanwhile, BUCL also outperforms MRPB and SWL-Best [21]. Furthermore, the minor hardware overhead and low-complexity architecture of BUCL make it easy to be implemented in hardware.

The rest of this paper is organized as the follows. Section Ⅱ introduces the baseline GPU architecture. Section Ⅲ presents an in-depth L1D performance analysis on memory loads with different memory un-coalescing degrees. The detailed cache bypassing scheme is discussed in Section Ⅳ. The experiment results are analyzed in Section V. Section Ⅵ addresses the difference of our scheme from the related work. Section Ⅶ concludes this paper.

II. GPU Background

This section provides background on GPU. Although the NVIDIA terminology is used in this paper, our approach is also suitable for any generic throughput-oriented multithreading processors.

A. CUDA Programming Model

A CUDA application is constituted of kernels executed on GPUs. In the same stream, kernel execution is sequential, and kernels from different streams could be concurrent. In a CUDA kernel, the threads are grouped into cooperative thread arrays (CTAs) or thread blocks (TBs), and a subgroup with 32 threads in a CTA form a warp. The number of threads in a CTA is a parameter being defined by the programmer, and a CTA usually contains enough threads to form multiple warps. Instructions are executed in a lockstep manner for all the threads in a warp. Many warps run in an interleaved manner for latency hiding.

B. Baseline GPU Architecture

Fig. 1 shows the baseline GPU architecture which is similar to the NVIDIA Fermi architecture. One GPU consists of multiple SM, and each SM adopts the single instruction multiple threads (SIMT) execution mode. An instruction is fetched and decoded for a group of threads constituting a warp. For the NVIDIA Fermi architecture, two warp schedulers manage all active warps in one SM. In every cycle, a warp scheduler selects one warp from all the ready warps to be issued to the computing units (ALUs or SFUs) or load/store unit (LD/ST). LD/ST may access data in L1D, shared memory, L1 texture cache (L1T), and L1 constant cache (L1C). L1D and shared memory are responsible to serve global memory and scratchpad accesses, respectively. L1D and shared memory share the same hardware resources, and their capacities could be configured.

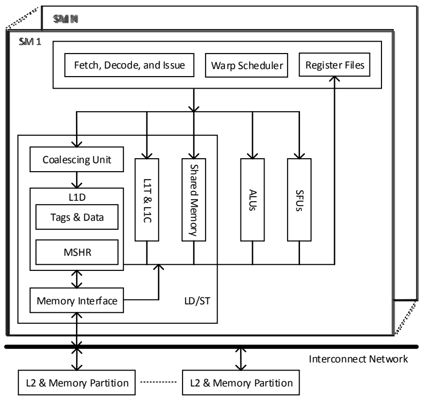

Figure. 1 Baseline GPU Architecture

When a global memory access instruction is issued, it is sent to the memory coalescing unit, which merges memory accesses of all the threads in a warp to minimize the number of memory transactions. If the data to be accessed for the 32 threads in a warp could be in one cache data block, the coalescing unit will generate just one access to L1D. Otherwise, multiple memory accesses are generated. The memory accesses from the coalescing unit are serviced by L1D sequentially. For a load L1D access, in the case of cache hit, the requested data is sent to the register file immediately. If a cache miss is encountered, the cache miss handing logic will check the miss status holding registers (MSHRs) to identify if the request is already pending for prior ones. If so, the request will be added into the waiting list in the same MSHR entry, and no new load request is issued to the lower levels of memory hierarchy. If not, a MSHR entry and cache block is allocated for this request, and at the same time, the load request is pushed into the miss queue. When any resource is depleted, e.g., there are no available MSHRs; all cache blocks in the set are reserved to be filled; or the miss queue is full, the miss handing logic could not service the new memory request and the memory pipeline stalls. The stalled request will retry to access L1D every cycle until the needed resources are available.

Each SM is connected to memory partitions through an interconnected network. A memory partition mainly consists of a L2 cache bank and a memory controller managing off-chip DRAM. The L2 cache is write back and write-allocate. L1D is write through with write-allocate or write-no-allocate [3, 4].

TALBE I. GPU Benchmarks (CUDA)

Name	Description	Num of Dynamic Insts	Suites
Memory Un-coalesced Benchmarks			
KMN	Kmeans Clustering	240945160	[21]
BFS	Breadth-First Search	460115486	[22]
PTF	Particle Filter	106432080	[22]
GAU	Gaussian Elimination	51392475	[22]
SSSP	Single-Source Shortest Paths	518911580	[5]
Memory Coherent Benchmarks			
BP	Back Propagation	190054784	[22]
BT	B+ Tree	542310058	[22]
NW	Needleman-Wunsch	207696896	[22]
2DC	2 Dimension Convolution	922103954	[23]
MYC	Myocyte	1006082	[22]

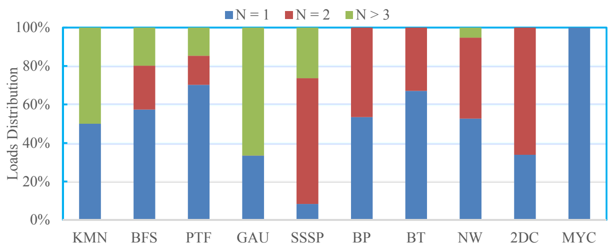

Figure 2. Distribution of load instructions with different memory un-coalescing degrees. N denotes the number of memory accesses for a load after memory coalescing.

III. Motivation

The effectiveness of L1D for loads with different memory un-coalescing degrees is evaluated in this section. A wide range of GPU applications are selected from Rodinia [22], CCWS [21], Polybench-GPU [23] and Lonestar suites [5]. Table I lists all the selected benchmarks, and the benchmarks run until completion or the performance becomes stable.

A. Benchmarks Categorization

These selected benchmarks are classified into the memory un-coalesced group and the memory coherent group according to the distribution of loads with different memory un-coalescing degrees, i.e., the number of memory requests per load after memory coalescing. The distribution of the loads with different numbers of memory accesses is showed in Fig 2. If the ratio of loads with memory un-coalescing degree more than two is bigger than 10%, we categorize this benchmark in the memory un-coalesced group. Otherwise, it is categorized into memory coherent group.

B. L1D Performance

The un-coalesced loads usually generate massive memory accesses. For instance, in the benchmark KMN, the loads producing 32 memory requests for a warp (i.e., memory un-coalescing degree of 32) take half of the total loads, but the proportion of these memory accesses generated by these loads is about 97% of all the memory accesses. Such massive accesses aggravate the cache contention, and cache trashing and pollution become frequent. As a result, the data reuse of cache is also poor as shown in Fig. 3, which reports the L1D

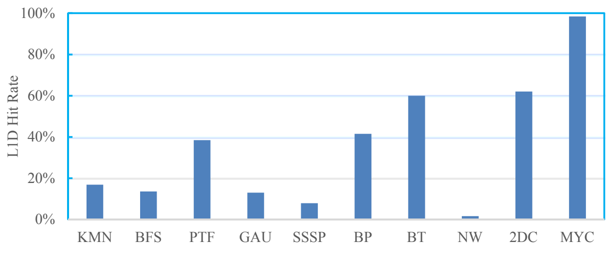

Figure 3. L1D hit rate

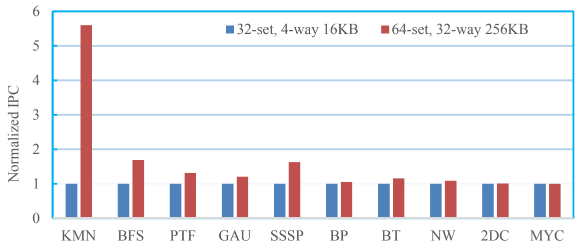

Figure 4. Performance improvement by increasing L1D capacity

hit rates for different benchmarks. From the figure, it is obvious that the L1D hit rate of coherent benchmarks is better than that of un-coalesced benchmarks except NW. The memory accesses in NW are mainly for shared memory, and the cache data fetched by global load warps are not reused.

In general, the performance of memory un-coalesced benchmarks is also more sensitive to cache capacity than memory coherent benchmarks, as shown in Fig. 4, because more cache capacity is needed to alleviate contention and make the data with good locality to be cached for longer time. Overall, we conclude that memory un-coalescing is an important reason for the low efficiency of L1D on GPUs.

C. Memory Divergence

For an un-coalesced load, if not all its memory accesses hit on L1D, the accesses missing on L1D will go to lower memory to acquire data. Such memory divergence makes the hits in L1D useless as the whole warp is stalled until the cache misses complete. In the case when all the accesses either uniformly miss or hit in the cache, there is no memory divergence. A load with all its un-coalesced accesses hit in the L1D is called an all-hit load (AHL).

Based on the un-coalescing degree, loads are categorized into five groups, and we analyze their performance characteristics in Fig. 5. In Fig. 5a, we show the AHL ratio for loads with different memory un-coalescing degrees. From the figure, we can see that the AHL ratio of loads with low degrees of memory un-coalescing is much higher than that of the loads with high degrees of memory un-coalescing. For instance, in the benchmark KMN, the AHL ratio for coherent/coalesced loads (i.e., the memory un-coalescing degree of 1) is 23%. In contrast, the AHL ratio of loads with memory un-coalescing degree of 21 to 32 is only 0.2%.

The loads except AHLs need to send memory requests to access memory partitions, and this delay is much longer, which is often hundreds of cycles. In Fig. 5b, we show the average latency from issue to completion normalized to the average latency of coherent loads. It could be observed that

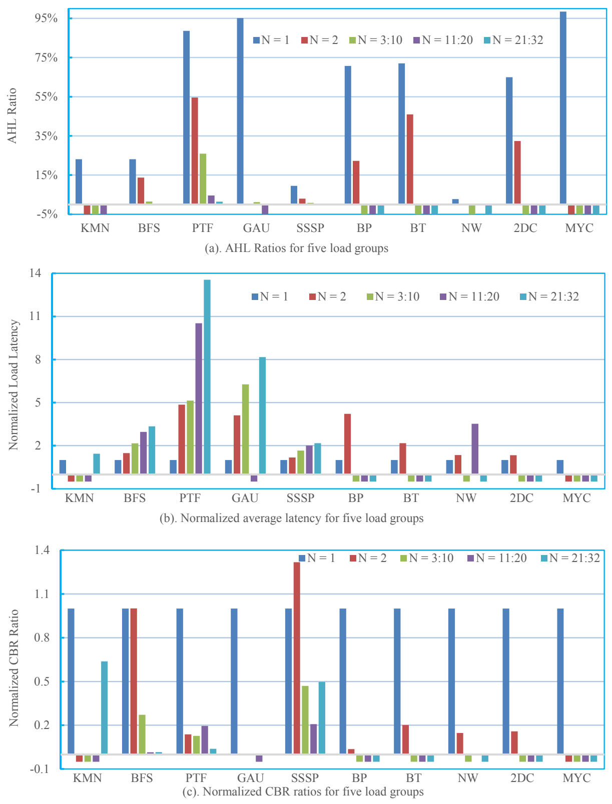

Figure 5. The effectiveness of L1D for load groups with different memory un-coalescing degrees. The loads are classified into five groups according to their memory un-coalescing degrees. The first group constitutes of the coherent loads, which have one memory access per load (i.e., the un-coalescing degree is 1). The other four groups include loads with 2, 3 to 10, 11 to 20, and 21 to 32 memory accesses per load, respectively. The negative values in the figure was used to highlight that there are no loads in this group.

when the memory un-coalescing degree is high, the average latency becomes much longer. For instance, in the benchmark BFS, the average latency of the loads with un-coalescing degrees of 21 to 32 is about 3.5 times of that of the coalesced loads. For the benchmark PTF, this ratio becomes 13.6.

Besides the AHL ratio and average latency, another important observation is that the probability of the cache block being reused also correlate to the memory un-coalescing degree of the demanding load. For a load, if the number of cache blocks requested by it is M (i.e., the un-coalescing degree), and the number of these cache blocks being reused is K, the ratio of K to M is the cache block reusing (CBR) ratio. The normalized CBR ratios for loads with different memory un-coalescing degrees are in Fig 5c. The CBR ratios are normalized to the CBR ratio of coherent loads. It could be seen that the cache blocks filled by coalesced loads are more likely to be reused, and the cache blocks filled by loads with high degrees of memory un-coalescing is much less likely to be reused. For instance, in the benchmark BFS, the normalized CBR ratios of loads with memory un-coalescing degrees of 11 to 20 and 21 to 32 are only 1.4% and 1.5%, respectively. This observation indicates that cache data filled by loads with low degrees of memory un-coalescing usually have better locality. More specifically, the cache data filled by loads with memory un-coalescing degrees lower than or equal to 2 are more likely to be reused.

In summary, the effectiveness of L1D for loads with different memory un-coalescing degrees show that it is unworthy to grant the L1D to the loads with severe un-coalescing, especially when there are massive memory accesses contending for a low L1D capacity.

D. *Unbalanced Workload on Memory Partitions*

In our GPU model, a memory partition consists of two memory sub-partitions and either sub-partition has an L2 cache bank. The two memory sub-partitions share a memory controller to access the off-chip DRAM. In general, balanced workload among all memory sub-partitions is beneficial to the overall performance. The utilization of the input buffer in a memory sub-partition, which is responsible for data communication through the interconnect network, is used to evaluate the utilization of the sub-partition. We observe that the utilization of different memory sub-partitions may be unbalanced in some benchmarks. In addition, the utilization of the same memory sub-partition may also dramatically change during the program execution.

IV. SELECTIVE CACHE BYPASSING

As discussed in Section III, the memory un-coalescing characteristics of loads have strong impact on L1D performance. Loads with low degrees of memory un-coalescing are more likely to be all-hit loads. In addition, the cache blocks filled by such loads are also more likely to be reused. Based on these observations, we propose selective cache bypassing for un-coalesced loads (BUCL). The key idea is to preserve the L1D capacity for loads with low memory un-coalescing degrees, which results in caching data with high locality. It consists of bypassing in two granularities, instruction-level bypassing and memory access level bypassing. Instruction-level bypassing means that all accesses generated from an un-coalesced load will bypass L1D. At the memory access level, each individual access of the threads in a warp is examined to see whether it will bypass L1D or not.

A. *Architecture*

The overall memory architecture with the proposed BUCL is shown in Fig. 6. In the figure, the parts highlighted with the gray shade constitute BUCL. The solid lines indicate data paths and the dashed lines indicate control signals of BUCL. The network interface constituted of a request queue and a response queue is responsible for transferring data between SMs and the interconnect network.

After coalescing in LD/ST, a load may need to access L1D. If the load is un-coalesced, the memory requests from this load access L1D sequentially. In BUCL, the load request is sent firstly to the load bypass logic. If the number of un-coalesced memory accesses of this load is bigger than the threshold of un-coalescing degree (TUCD), the accesses generated from this load for all 32 threads in the warp will

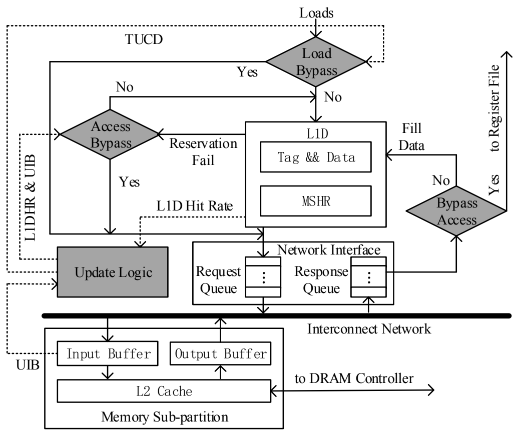

Figure 6. The memory architecture with the proposed BUCL.

bypass L1D and be inserted into the request queue to access the lower memory hierarchy. Otherwise, the requests of the load will try to access L1D.

When a memory request accesses L1D, the response may be a hit, a miss or a reservation fail. For a cache hit or miss, the memory request could be processed by L1D. A reservation fail means that L1D is congested and stalled, so the memory request could not be processed. In the baseline GPU, in the case of a reservation fail, the memory request will wait until the L1D is not stalled, which causes the memory pipeline to be stalled even the interconnect network and lower memory hierarchy are under-utilized. In BUCL, in the case of a reservation fail, the memory request is sent to the access bypass logic. If the L1D hit rate (L1DHR) and the utilization of the input buffer (UIB) at the corresponding memory sub-partition are smaller than the L1DHR-threshold and the UIB-threshold, and the request queue is not full, then this memory request will bypass L1D and be sent to the request queue. The L1DHR and UIB are the average L1D hit rate and average utilization of the input buffer in the target memory sub-partition during the previous sampling period, respectively.

BUCL adopts different bypassing granularities in the memory pipeline to improve the L1D efficiency. Before the load accesses L1D, the bypassing granularity is a load instruction. The instruction-level bypassing reduces the memory accesses pressure on L1D, and could also preserve the insufficient L1D resource for the cache data with high locality. After the memory request accesses L1D, the bypassing granularity becomes a memory access. Based on the L1DHR and UIB, the bypassing decision is made. The L1DHR indicates the locality of data fetched during the previous sampling period, and is used to predict whether the fetched data by the current memory access is likely to be reused. If the data locality is not considered, the memory access level bypassing behaves similar to MRPB and more cache misses may be incurred for subsequent memory accesses. The reason is that the data fetched by the bypassing accesses from lower memory will not be filled into L1D and be sent to register file directly. If the data with good locality are not cached, more latency would be spent on fetching the needed data, which hurts the overall performance. Furthermore, even the data being fetched are predicted to have poor locality, but the utilization of the memory sub-partition is high, the memory access level bypassing should be also avoided. The reason is that memory accesses on the memory sub-partition with high utilization may cause congestion, and as a result, the response will not be timely.

B. *Profiling Strategy and Parameter Updates*

There are five parameters, TUCD, L1DHR-threshold, UIB-threshold, L1DHR and UIB, used in BUCL for making bypassing decisions. Among these five parameters, L1DHR-threshold and UIB-threshold are defined by the programmers, and also fixed during GPU kernel execution. L1DHR and UIB are the average L1D hit rate and input buffer utilization collected during one sampling period, and updated at the end of every sampling period. L1DHR and UIB are stored in the update logic of BUCL, and used for the memory access granularity bypassing decision in the access bypass logic of BUCL.

It is challenging to determine the TUCD for instruction-level bypassing, which may be difficult for programmers. Therefore, we resort to hardware-based dynamic approaches, and two approaches are attempted. The first one is to leverage the multiple SMs in a GPU. At the beginning of kernel execution, SMs are configured with different TUCDs. After the performance profiling period, the TUCD in the SM, which obtains the highest performance, is set to be the TUCD to be used by all the SMs for subsequent execution. This method is referred to as BUCL-Prf. The second one is to update TUCD dynamically at the end of each sampling period. If the L1DHR of sampling period is bigger than L1DHR-threshold, TUCD is increased by one; otherwise, TUCD is decreased by one. Then TUCD is stored in the update logic to be used in the next sampling period. This approach is called BUCL-Dyn.

V. Experimental Results

We use GPGPU-Sim [24] v3.2.2, a cycle-accurate GPU microarchitecture simulator, to evaluate our proposed BUCL scheme. The architecture configuration is summarized in Table Ⅱ. A hashing function with a 5-bits XOR operation [25] is used as the cache set indexing method in both L1 and L2 to enhance the cache performance. We also model MRPB and SWL-Best to compare with BUCL. In MRPB, when there is any resource congestion that causes L1D to be stalled, the load memory access will bypass L1D until resources are available. SWL-Best is the static warp limiting (SWL) with the optimal configuration, and could outperform the dynamic cache conscious warp scheduling (CCWS) [21].

The BUCL configuration is also shown in Table Ⅱ. The sampling period is important, because long sampling period could not reflect the execution status timely while short sampling period may be sensitive to noises. We found that a sampling of 1000 cycles to be effective and it is used in our experiments. For BUCL-Prf, different SMs use different TUCDs during the profiling cycles. Both 5000 cycles and 10000 cycles are used as profiling periods for performance

TABLE Ⅱ. Architecture Configuration

Baseline Configuration	
SM Configuration	15 Cores, SIMD width=32, 1.4GHz 5-stage pipeline, max threads per core: 1536
L1 Data Cache/SM	16KB, 128B line, 4-wary assoc, 32 MSHR entries
Shared Memory/SM	48KB, 32 banks
Warp scheduling	GTO
L2 Unified Cache	768KB, 128b line, 6 banks
Cache Indexing	Hashing Function with XOR Operation [25]
DRAM	6 memory channels, FR-FCFS scheduler, BW: 8 bytes/cycle per channel,
GDDR5 Timing	t_{CL}=12, t_{RP}=12, t_{RC}=40, t_{RAS}=28, t_{RCD}=12, t_{WR}=12, t_{RRD}=6, t_{CDLR}=5
BUCL Configuration	
Sampling Period	1000 cycles
Profiling Cycles	5000/10000 cycles
L1DHR-threshold	0.8
UIB-threshold	0.7
Initial TUCD	5
TUCDs for Profiling <SM_ID, TUCD>	<1, 2><2, 3><3, 4><4, 5><5, 6> <6, 7><7, 8><8, 10><9, 12><10, 14> <11,16><12, 18><13, 20><14, 22><15, 25>

estimation. With 15 SMs, the initial TUCDs are set for a wide range from 2 to 25. For BUCL-Dyn, the initial value of TUCD is set to be 5. The maximum and minimum of TUCD are 25 and 2, respectively.

A. *Performance Comparison*

We evaluate the performance using the metric, instructions per cycle (IPC). Fig. 7 reports the performance of both memory un-coalesced benchmarks and memory coherent benchmarks, normalized to the baseline GPU. For memory un-coalesced benchmarks, MRPB, BUCL-Prf5k (5000 cycles for profiling), BUCL-Prf10k (10000 cycles for profiling), SWL-Best and BUCL-Dyn achieve an average of 22%, 38%, 29%, 16% and 36% IPC improvement compared to the baseline GPU, respectively. It is obvious that both BUCL-Dyn and BUCL-Prf significantly outperform MRPB and SWL-Best on average. In addition, if the degree of memory un-coalescing of a benchmark is high, both BUCL-Dyn and BUCL-Prf become more effective. For instance, in the benchmark KMN, the un-coalesced loads with 32 memory requests almost account for half of the total loads, the IPC improvement over MRPB is about 19%. But it is observed that BUCL-Prf is sensitive to the profiling cycles. For instance, in the benchmark GAU, although BUCL-Prf5k obtains the best IPC improvement, almost no performance improvement is acquired by BUCL-Prf10k. The reason for this situation is that the memory access pattern during the profiling cycles may not be representative for the memory characteristics of the overall kernel execution. If the profiling cycles are not proper, the performance would not be effectively improved. Therefore, BUCL-Dyn can be more suitable and effective than BUCL-Prf, although BUCL-Prf5k outperforms BUCL–Dyn dramatically in the benchmark GAU due to the learning process of BUCL-Dyn.

For memory coherent benchmarks, BUCL-Dyn and SWL-Best obtain 5% and 3% IPC improvement over the baseline

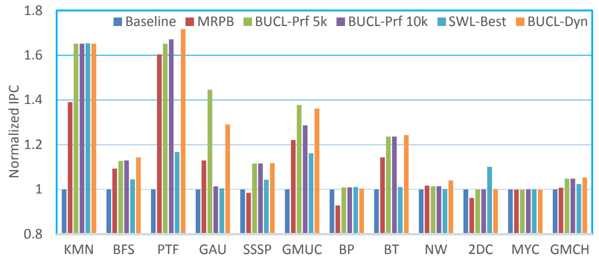

Figure 7. Normalized IPC Improvement, GMUC is the geometric mean of normalized IPC improvement for un-coalesced benchmarks, and GMCH is the geometric mean of normalized IPC improvement for coherent benchmarks

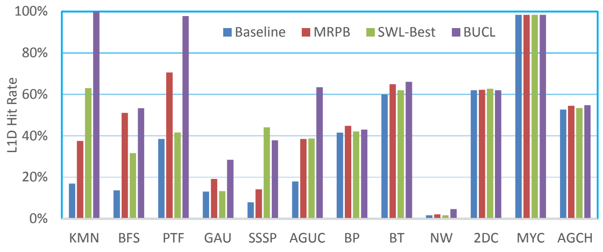

Figure 8. L1D hit rate, this is the average hit rate for the whole lifetime of benchmark. AGUC is the average L1D hit rate for memory un-coalesced benchmarks, and AGCH is the average L1D hit rate for memory coherent benchmarks.

architecture on average, respectively. Coherent benchmarks usually have regular memory references. Therefore, bypassing when cache is stalled used in MRPB may disturb the cache data locality. In Fig. 7, performance loss could be observed for MRPB, for benchmark BP and 2DC. Their IPCs drop by 7% and 4%, respectively, compared to the baseline architecture. On the contrary, such performance loss is not observed in BUCL-Dyn and BUCL-Prf, which indicates that our prediction of data locality based on the L1DHR is effective.

Overall, BUCL-Dyn outperforms MRPB and SWL-Best for both memory un-coalesced benchmarks and memory coherent benchmarks, and it also avoids the invalidity caused by improper profiling cycle for some un-coalesced benchmarks, which shows that BUCL-Dyn is an effective cache bypassing scheme to improve the GPU performance. In the following, BUCL-Dyn is referred as BUCL for simplicity.

B. *L1D Hit Rate Improvement*

The average L1D hit rates for different benchmarks are shown in Fig. 8. For memory un-coalesced benchmarks, the average L1D hit rates of baseline, MRPB, SWL-Best and BUCL are 18%, 39%, 39% and 63%, respectively. It confirms that BUCL improves the L1D hit rate significantly for memory un-coalesced benchmarks. For memory coherent benchmarks, BUCL improves the average hit rate slightly. It is observed that for the benchmark SSSP, SWL-Best obtains higher L1D hit rate over BUCL although BUCL obtains better IPC. The similar phenomenon could be also observed in the benchmark BP, MRPB loses performance, but obtains higher L1D hit rate compared to other approaches. This indicates that high L1D rate does not always result in high

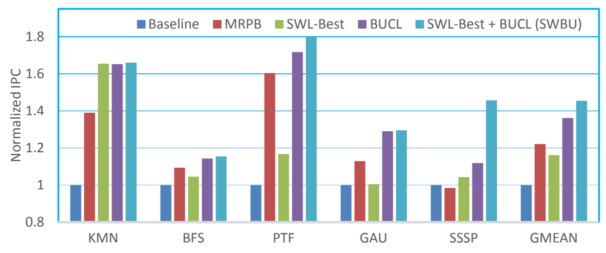

Figure 9. Normalized IPC, GMEAN is the geometric average of normalized IPC of un-coalesced benchmarks.

performance on GPUs.

C. *Design Exploration*

Warp throttling is effective to improve the efficiency of L1D and the GPU performance for cache sensitive benchmarks [21, 26]. Memory un-coalesced benchmarks in this paper are also cache sensitive as the discussion in the motivation section. If warp throttling is adopted, although the warp number is limited, un-coalesced loads may be still existing. So it may be possible to further improve the performance with BUCL. To verify the feasibility of this idea, the combination of SWL-Best and BUCL, which is referred as SWBU, is evaluated, and the normalized IPCs are shown in the Fig. 9. We could see that SWBU obtains 45% IPC improvement over baseline on average for un-coalesced benchmarks, and SWBU outperforms both SWL-Best and BUCL. This indicates that BUCL is orthogonal to the warp throttling scheme. In addition, SWBU is also beneficial to improve the concurrency, which is shown in Table III. For un-coalesced benchmarks, all the optimal warp numbers picked by SWBU become bigger. So, warp throttling and BUCL could be complementary to reduce L1D contention and improve the performance and concurrency on GPUs.

D. *Hardware Cost*

Compared to prior GPU cache bypassing schemes, BUCL is easy to be implemented, and does not need any significant on-chip memory. As the memory access patterns are nearly the same among different SMs, only one L1D is used to track the hit rate. The update logic is also shared among all SMs. In the update logic, when updating at the end of one period, some logic resources are required to update and record the UIBs of different memory sub-partitions, the L1DHR and TUCD, respectively. In addition, as the memory partition and SMs are usually in different clock domains, the asynchronous transfer logic across clock domains is necessary to send UIBs to update logic. Overall, the hardware overhead for BUCL is minor, and could be negligible compared with the whole GPU.

VI. Related Work

There are many proposals for CPU cache bypassing, such as [27, 28], but these CPU-based bypass schemes are usually for the last level caches (LLCs). The accesses to LLCs are filtered by previous caches, and the data locality of LLCs is not as poor as that of L1D in GPUs. So these approaches is difficult to improve the GPU performance, and we mainly review the works on GPUs.

Table III The optimal warp numbers for SWL-Best and SWBU

Benchmarks	SWL-Best	SWBU
KMN	3	12
BFS	12	18
PTF	20	41
GAU	7	14
SSSP	5	10

Bypassing L1D is adopted in multiple approaches to improve the cache efficiency and GPU performance. MRPB [14] proposes FIFOs to reorder memory accesses for reducing warp contention, and it also bypasses L1D once L1D is stalled because of unavailable resources. MRPB does not consider the data locality as evaluated in Section V, which will cause the data with strong locality not to be cached in L1D. Based on the locality and reuse distance of GPGPU programs, load warps are divided into cached, bypassed and waiting in [18]. A locality-driven dynamic bypassing design is proposed in [11], only data with high reuse and short reuse distance could be stored in L1D. Another work on cache bypassing based on reuse distance is proposed in [15]. If the fetched data is predicted to be zero-reuse, it will bypass L1D and is sent to compute units directly. All the bypassing schemes introduced by [11, 15, 18] are based on the data locality, but on-chip memory resources are required to track the locality, and modifying the cache structure is also needed [11]. The L2 cache in memory partitions is enhanced to track the access history [13]. If a block in the L2 cache is requested twice by the same SM, it indicates that severe contention occurs in L1D, and all the accesses to the target set will be bypassed. L2 is also used to detect locality in [12], both bypassing cache and warps throttling are adopted in this work.

Complier based bypassing techniques are also investigated [16, 17] to improve the GPU performance, but the static compiler based bypassing mainly work for regular workloads. For the irregular programs especially with memory un-coalescing, compiler-level analysis can be highly challenging. None of above bypassing schemes target at memory un-coalescing. It is observed that warps tend to exhibit stable cache hit behavior over long periods of execution in [20], and cache bypassing and insertion policies are also proposed for the L2 cache. Based on the combination of warp's priority and memory divergence, L1D management policy is introduced in [19], which mainly focuses on insertion and replacement, and also needs extra victim cache to track warp locality. Compared to these methods, BUCL could reduce L1D contention effectively, and meanwhile, it does not need extra precious on-chip memory resources and is also easy to be implemented.

Warp scheduling plays an important role on generating the memory access pattern in GPU, and previous works on warp scheduling try to improve cache efficiency, such as [21, 26]. CCWS is proposed in [21], when locality loss is detected, CCWS limits the number of load warps that could be issued. Divergence-aware warp scheduling (DAWS) [26] uses cache usage predications to schedule warps, such that data reused by active threads is unlikely to exceed the L1D capacity. Both CCWS and DAWS are warp throttling methods to alleviate the contention on GPU L1D. BUCL is orthogonal to these warp scheduling schemes, because in the reduced concurrency, the un-coalescing loads still exist, and BUCL could be used to further improve the cache efficiency. Similarly, BUCL is orthogonal to warp-level bypassing approaches [29, 30], as the warps chosen to access L1D can also contain un-coalesced loads, reducing the efficacy of L1D.

VII. Conclusions

In this paper, the L1D effectiveness for loads with different memory un-coalescing degrees is analyzed, and a simple yet effective GPU cache bypassing scheme, BUCL, is proposed. It is easy to be implemented and has very minor hardware overhead. Our experimental results show that BUCL obtains an average of 36% and 5% IPC improvement over GPU baseline architecture for memory un-coalesced and memory coherent benchmarks, respectively, and also significantly outperforms MRPB and SWL-Best.

Acknowledgment

This work is founded by National Natural Science Foundation of China (grant number is 61231018 and 61273366), and a Chinese research program "introducing talents of discipline to universities B13043".

References

[1] N. Corporation, *CUDA C Programming Guide v5.5*. 2013.

[2] K.O.W. Group, *The OpenCL C Specification Version: 2.0*. 2013.

[3] NVIDIA, *NVIDIA's Next Generation CUDA Compute Architecture: Fermi*. 2009.

[4] NVIDIA, *NVIDIA's Next Generation CUDA Compute Architecture: Kepler GK110*. 2012.

[5] M. Burtscher, R. Nasre, and K. Pingali. *A quantitative study of irregular programs on GPUs*. in *Workload Characterization (IISWC), 2012 IEEE International Symposium on*. 2012.

[6] M.A.O. Neil and M. Burtscher. *Microarchitectural performance characterization of irregular GPU kernels*. in *Workload Characterization (IISWC), 2014 IEEE International Symposium on*. 2014.

[7] N. Fauzia, L.-N. Pouchet, and P. Sadayappan, *Characterizing and enhancing global memory data coalescing on GPUs*, in *Proceedings of the 13th Annual IEEE/ACM International Symposium on Code Generation and Optimization*. 2015, IEEE Computer Society: San Francisco, California. p. 12-22.

[8] B. Wu, Z. Zhao, E.Z. Zhang, Y. Jiang, and X. Shen, *Complexity analysis and algorithm design for reorganizing data to minimize non-coalesced memory accesses on GPU*, in *Proceedings of the 18th ACM SIGPLAN symposium on Principles and practice of parallel programming*. 2013, ACM: Shenzhen, China. p. 57-68.

[9] A. Jaleel, K.B. Theobald, J. Simon C. Steely, and J. Emer, *High performance cache replacement using re-reference interval prediction (RRIP)*, in *Proceedings of the 37th annual international symposium on Computer architecture*. 2010, ACM: Saint-Malo, France. p. 60-71.

[10] M.K. Qureshi, A. Jaleel, Y.N. Patt, S.C. Steely, and J. Emer, *Adaptive insertion policies for high performance caching*, in *Proceedings of the 34th annual international symposium on Computer architecture*. 2007, ACM: San Diego, California, USA. p. 381-391.

[11] C. Li, S.L. Song, H. Dai, A. Sidelnik, S.K.S. Hari, et al., *Locality-Driven Dynamic GPU Cache Bypassing*, in *Proceedings of the 29th ACM on International Conference on Supercomputing*. 2015, ACM: Newport Beach, California, USA. p. 67-77.

[12] X. Chen, L.W. Chang, C.I. Rodrigues, J. Lv, Z. Wang, et al. *Adaptive Cache Management for Energy-Efficient GPU Computing*. in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*. 2014.

[13] X. Chen, S. Wu, L.-W. Chang, W.-S. Huang, C. Pearson, et al., *Adaptive Cache Bypass and Insertion for Many-core Accelerators*, in *Proceedings of International Workshop on Manycore Embedded Systems*. 2014, ACM: Minneapolis, MN, USA. p. 1-8.

[14] W. Jia, K.A. Shaw, and M. Martonosi. *MRPB: Memory request prioritization for massively parallel processors*. in *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*. 2014.

[15] Y. Tian, S. Puthoor, J.L. Greathouse, B.M. Beckmann, D.A. Jim, et al., *Adaptive GPU cache bypassing*, in *Proceedings of the 8th Workshop on General Purpose Processing using GPUs*. 2015, ACM: San Francisco, CA, USA. p. 25-35.

[16] X. Xie, Y. Liang, G. Sun, and D. Chen. *An efficient compiler framework for cache bypassing on GPUs*. in *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 2013.

[17] X. Xie, Y. Liang, Y. Wang, G. Sun, and T. Wang. *Coordinated static and dynamic cache bypassing for GPUs*. in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. 2015.

[18] Z. Zheng, Z. Wang, and M. Lipasti, *Adaptive Cache and Concurrency Allocation on GPGPUs*. IEEE Computer Architecture Letters, 2015. 14(2): p. 90-93.

[19] B. Wang, W. Yu, X.-H. Sun, and X. Wang, *DaCache: Memory Divergence-Aware GPU Cache Management*, in *Proceedings of the 29th ACM on International Conference on Supercomputing*. 2015, ACM: Newport Beach, California, USA. p. 89-98.

[20] R. Ausavarungnirun, S. Ghose, O. Kayiran, G.H. Loh, C.R. Das, et al. *Exploiting Inter-Warp Heterogeneity to Improve GPGPU Performance*. in *2015 International Conference on Parallel Architecture and Compilation (PACT)*. 2015.

[21] T.G. Rogers, M. O'Connor, and T.M. Aamodt. *Cache-Conscious Wavefront Scheduling*. in *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*. 2012.

[22] S. Che, M. Boyer, J. Meng, D. Tarjan, J.W. Sheaffer, et al. *Rodinia: A benchmark suite for heterogeneous computing*. in *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*. 2009.

[23] S. Grauer-Gray, L. Xu, R. Searles, S. Ayalasomayajula, and J. Cavazos. *Auto-tuning a high-level language targeted to GPU codes*. in *Innovative Parallel Computing (InPar), 2012*. 2012.

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

[25] C. Nugteren, G.J.v.d. Braak, H. Corporaal, and H. Bal. *A detailed GPU cache model based on reuse distance theory*. in *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*. 2014.

[26] T.G. Rogers, M. O'Connor, and T.M. Aamodt, *Divergence-aware warp scheduling*, in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*. 2013, ACM: Davis, California. p. 99-110.

[27] N. Duong, D. Zhao, T. Kim, R. Cammarota, M. Valero, et al. *Improving Cache Management Policies Using Dynamic Reuse Distances*. in *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*. 2012.

[28] J. Gaur, M. Chaudhuri, and S. Subramoney. *Bypass and insertion algorithms for exclusive last-level caches*. in *Computer Architecture (ISCA), 2011 38th Annual International Symposium on*. 2011.

[29] H. Dai, C. Li, H. Zhou, S. Gupta, C. Kartsaklis, et al., *A model-driven approach to warp/thread-block level GPU cache bypassing*, in *Proceedings of the 53rd Annual Design Automation Conference*. 2016, ACM: Austin, Texas. p. 1-6.

[30] D. Li, M. Rhu, D.R. Johnson, M. O'Connor, M. Erez, et al. *Priority-based cache allocation in throughput processors*. in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. 2015.