APRES : Improving Cache Efficiency by Exploiting Load Characteristics on GPUs

Anuj Dubey, Akshay Singh, Divyansh Jagota, Sopan Patra

***Abstract*— Performance of GPGPU applications is severely impacted by long-latency memory instructions. Interleaving warp execution can not hide such long-latencies ranging hundreds of cycle. Also, Thread-Level Parallelism at a large-scale degrades the benefits of cache hierarchy by introducing cache contention. This paper proposes a novel technique to improve GPU cache-efficiency.**

***Keywords*-GPGPU;Warp Scheduling;Prefetching**

I. INTRODUCTION

Modern day GPUs aim to exploit Thread Level Parallelism in order to achieve high throughput. However, long off-chip memory access latency is difficult to hide and remains exposed despite warp-execution interleaving. Clearly, memory system performance influences GPU performance. Small L1 data cache size leads to a high miss rate in GPUs. Figure1 highlights that Capacity and Conflict misses are the dominating misses in memory-intensive applications and account for about 63% of the miss rate.A hypothetical GPU model with a very large L1 cache (size 32 MB) cuts Capacity and Confict misses to almost half which leads to a significance improvement in GPU performance. But such large cache designs are impractical in current design methodologies. On the other hand, strong strided pattern of loads in GPUs introduces opportunities for prefetching which helps combat cold misses. But small cache size in GPUs makes prefetching a challenging task due to early eviction of prefetched blocks. So there is a need to take advantage of cache access pattern in static loads while trying to minimize the rate of early evictions.

II. RELATED WORK

A. Warp Scheduling Techniques

Rogers et al. proposed a Cache-Conscious Wavefront Scheduling (CCWS) to leverage the intra-warp locality for efficient cache utilization. They also designed another warp scheduling method which focuses on prediction of cache utilization during memory divergence. Sethia et al. proposed a warp scheduling and a cache re-execution method for the efficient utilization of memory hardware resource to improve the memory-intensive workloads. Lee et al. recently introduced two warp scheduling techniques to improve the performance balance between faster and slower warps, which are critical to the overall performance.

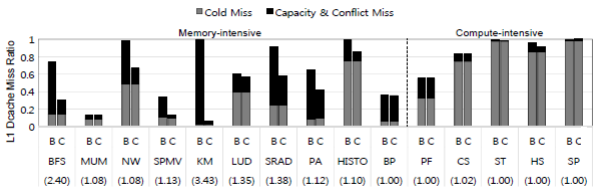

Fig. 1. Breakdown of total L1 data cache miss ratio, capacity and conflict miss ratio of Baseline 32KB (B) and large 32MB Capacity of L1 data cache (C) (Number in parentheses: Relative performance of C normalized to B).

B. Prefetching

Lee et al. proposed a many-thread aware prefetching scheme considering massively parallel processing on GPUs. The scheme consists of the inter-thread prefetching and per-warp stride prefetching. Also, Sethia et al. introduced a new approach of per-warp stride prefetching to reduce the power consumption of GPUs. It helps to save the energy by reducing the number of active warps. To address the poor performance of irregular memory access, Lakshminarayana et al. designed a prefetching scheme which is especially focused on the graph applications. The methodology utilized the spare register file to store the prefetched data.

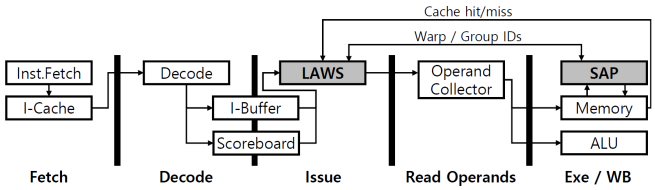

Fig. 2. GPU architecture with APRES

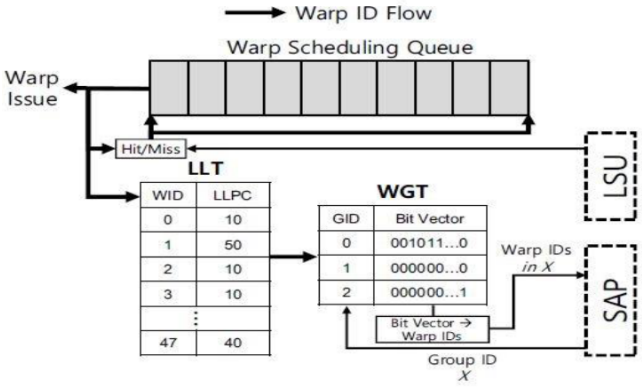

Fig. 3. LAWS architecture

Moreover, it focused to exploit the invalid thread operations by changing the valid load instruction executions.

C. Combined Warp Scheduling and Prefetching

Recently, combined warp scheduling and prefetching scheme was proposed to improve GPU performance. By scheduling warp groups while considering the spatial locality to reduce the stalls from memory pipeline leads to improved performance. They could operate the prefetching with little overhead of memory traffic. However, the introduced prefetching techniques did not show significant performance improvement because they did not address early evictions of prefetched data. Besides, it is found that more than 60% of late prefetching still exists in their experimental results.

III. Design

The proposed technique, APRES, cooperatively operates warp scheduling with prefetching to improve cache efficiency. Figure 2 illustrates additional two modules added to the GPU pipieline - *Locality Aware Scheduling* (LAWS), *Schedule Aware Prefetching* (SAP).

A. Locality Aware Scheduling (LAWS) module

There are two additional structures needed by the LAWS module: The Last Load Table(LLT) and the Warp Group Table(WGT). Figure 3 illustrates the LAWS architecture. When a warp issues a load, it stores its load PC in the corresponding LLPC field of the LLT. When that warp issues a different load PC, all the warp entries in the LLT check if their LLPC match the LLPC of the issued warp. If there is a match, a new warp group is formed and a bit vector is stored in the WGT. In execute stage, if the warp hits in the cache, the warps of the corresponding group are pushed to the head of the scheduling queue. If the load misses, the warps of the group are pushed to the tail of the scheduling queue. Since the formation of the groups happens in issue stage and the cache access result is known in the execute stage, we need to buffer three warp group bit vectors for the three intermediate stages.

B. Schedule Aware Prefetching (SAP) module

When a demand request misses in the cache for a warp, LAWS module sends all the warp IDs present in the group corresponding to that warp to SAP module for prefetching. SAP consists of a Prefetch Table(PT), a Warp Queue(WQ) and a Demand Request Queue(DQ). All missed demand requests from LAWS are queued in DQ. Each demand request contains Warp ID, PC and the address of the load which just missed in the cache. PT contains the history of previous load instructions. It is PC indexed and consists of Warp ID, Stride and Address for all the previous accesses. WQ contains all the Warp IDs for the group which is currently serviced.

Figure 3 illustrates the SAP architecture. PC for the current demand request is searched in PT. If the PC is found, a new stride is calculated by dividing the difference of the addresses in PT and Demand Request by the difference of Warp IDs in PT and demand. If the currently calculated stride matches the stride already present in PT, strided access pattern is implied for the loads of that group. Clearly, prefetching should be done. Prefetch addresses are generated for each Warp ID

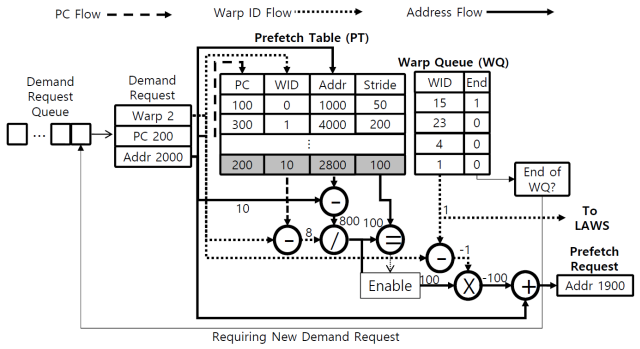

Fig. 4. SAP architecture

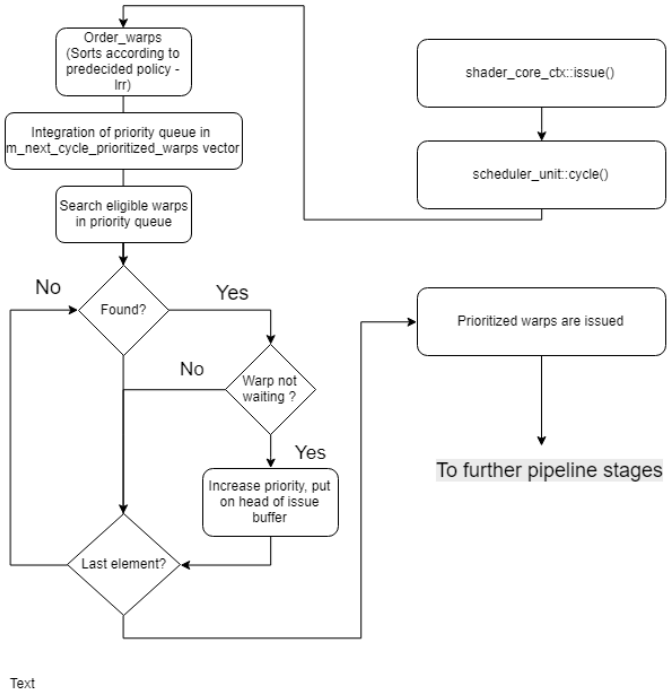

Fig. 5. Updating Scheduling Queue priority using LAWS

present in WQ using the currently calculated stride and address present in the demand request. ALso, these Warp IDs are sent back to LAWS so that they can be prioritized for scheduling in order to avoid early eviction. If PC of the demand request does not match any PC present in PT, then a new entry is pushed into PT. If currently calculated stride does not match already present stride, prefetching is disabled and PT is updated.

IV. Methodology

A. Locality Aware Scheduling (LAWS) module

- An object of the class LAWS has been added to each core of the gpgpu-sim. The class LAWS consists of Last Load Table(LLT), Warp Group Table(WGT) and a priority queue.
- LLT table is an array of last_load_table_entry type which is a structure containing last load PC and a valid bit associated with each entry.
- The WGT table is a table of three entries denoting the groups associated with the warps in the issue, operand collector and the execute stage respectively. It has an unsigned int bit vector and a boolean valid bit associated with it.
- The update priority function has been overloaded for the two different stages of the pipeline i.e. the issue stage and the execute stage.
- In the issue stage the update priority function forms the group based on the LLT table and updates the bit vector in the WGT table. This group identity is also returned to the warp inst object to make it flow down the pipeline along with the warp.
- In the execute stage the update priority function based upon the hit or miss in cache moves the corresponding warps in the group are moved to the head or the tail of the priority queue. In case of a miss all this information is sent to the SAP module for prefetching.

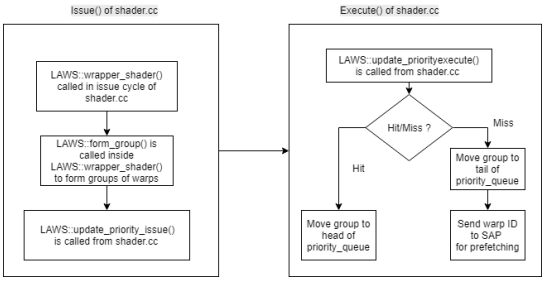

Fig. 6. LAWS implementation flowchart

B. Schedule Aware Prefetching (SAP) module

- In order to implement the functionality of SAP module, sap class is added to gpgpu-sim. It consists of two vectors prefetch_table (PT) and warp_queue (WQ). prefetch_table is a vector of type ptable_entry which is a struct that consists of PC, WarpID, Address and Stride.

- process_memory_access_queue function of ldst_unit class in shader.cc provides hit/miss status when L1D cache is accessed by a warp. Upon a miss, warp_queue (WQ) should be populated with the Warp IDs of all other warps of that group. Bit vector representing that group is got from get_group_ID function in laws class. WIDs are extracted from the bit vector and pushed into warp_queue vector.
- Next, search_pc function in sap class is invoked to find if the PC of the demand request is already present in prefetch_table (PT). If it is present, the corresponding row in PT is returned which is of type ptable_entry.
- Using the address and WarpId present in this struct returned and the address and WarpId of demand request, a new stride is calculated using the calculate_stride function in sap class.
- compare_stride function in sap class is used to compare the currently calculated stride and the stride present in prefetch_table (PT). If it is a match, prefetch addresses are generated for each WarpId in warp_queue (WQ) using the generate_prefetch_request function in sap class.
- After this, prefetch_table (PT) is updated using update_prefetch_table function in sap class. If there was a match for the PC in demand request and the PC in PT, values in PT are simply updated. Else, a new row is pushed into PT.

V. Results & Future Work

GPGPU-sim v3.2.2 is used for implementing the idea. Also, power model from GPUWattch is used. Parboil benchhmark suite and Rodinia benchmark suite were used from CUDA software development kit. Figure 8 and Figure 9 illustrate the change in IPC for Baseline GPGPU architecture and the proposed GPGPU architecture for the aforementioned benchmarks.

For BFS workload, an improvement of 38.6% is seen in IPC. For srad_v1, LAWS degraded the performance by 0.4% and for srad_v2 benchmarks, LAWS improved the performance by 4.4%. We also ran the parboil benchmark on LAWS. For SPMV workload, a 20% improvement in IPC is observed. This result exactly matches with what

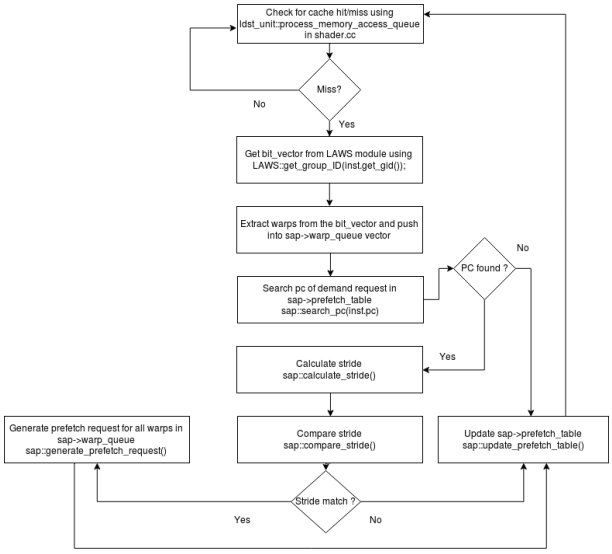

Fig. 7. SAP implementation flowchart

TABLE I
Normalized IPC Comparison - Baseline Vs Rodinia Benchmarks

Workload	LAWS
srad_v1	0.996
srad_v2	1.044
bfs	1.386

is claimed by the authors of this paper . IPC improvement for histo, lbm, mri_gridding and sgemn are 1%,2.3%,7% and 0.5% respectively. mri-q and stencil benchmarks show degradation of 1.2% and 1.4% respectively . Table I and Table II highlight the normalized IPC when LAWS is integrated for Rodinia and Parboil benchmark suites respectively.

We were able to implement LAWS module successfully into the baseline GPGPU architecture. We also implemented SAP module and were able to generate prefetch requests for Warp IDs in case the head warp of that group suffered a miss in the cache. We are currently working on sending the generated prefetch requests to LSU unit and the prefetched Warp IDs back to the LAWS module so that they can be prioritized for scheduling in order to avoid early evictions. We can come up with statistics for Cold misses in L1D cache after we are able to achieve that.

TABLE II
NORMALIZED IPC COMPARISON - BASELINE VS PARBOIL BENCHMARKS

Workload	LAWS
histo	1.014
lbm	1.023
mri_gridding	1.070
mri_q	0.988
sgemm	1.005
spmv	1.190
stencil	0.986

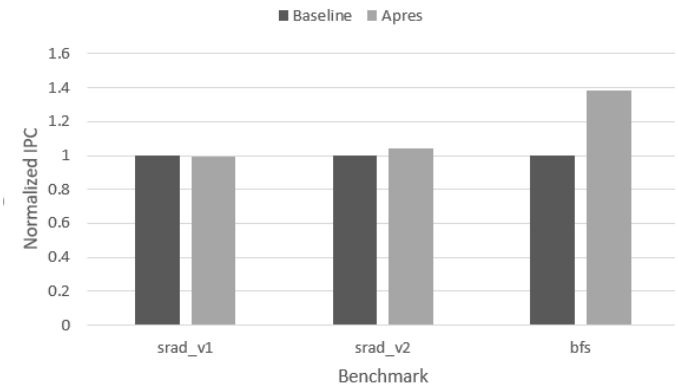

Fig. 8. IPC comparison for Rodinia benchmarks (A) srad_v1 (B) srad_v2 (C) bfs

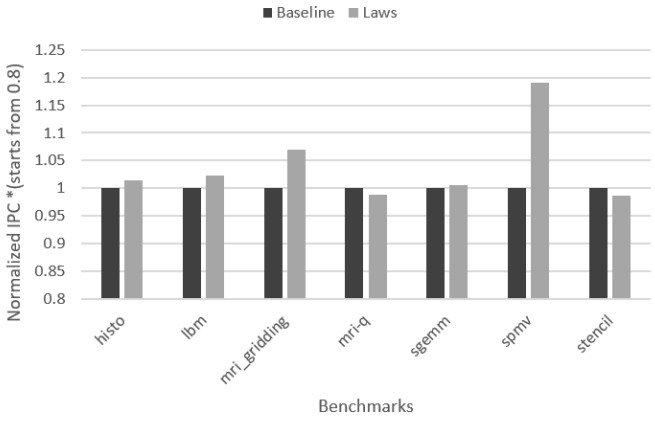

Fig. 9. IPC comparison for Parboil benchmarks (A) histo (B) lbm (C) mri_gridding (D) mri-q (E) sgemm (F) spmv (G) stencil

VI. CONCLUSION

Through our experiments and the subsequent results, we can confirm that cache access behavior of static GPU loads exhibit strong locality while other loads have a large memory footprint but with a stride access pattern. These characteristics are unique to each static load regardless of warp IDs. APRES tries to exploit the above observations. APRES architecture tries to combine locality-aware warp scheduling and a inter-warp prefetching . The LAWS warp scheduler tries to group warps such that all the warps in the group execute a particular load instruction within a short window of time. Thus it enables multiple consecutive accesses to a cache line before it gets evicted. APRES, for certain benchmarks, achieves significant performance improvement over any combinations of existing warp scheduling and prefetching techniques.

REFERENCES

[1] Yunho Oh , Keunsoo Kim, Myung Kuk Yoon, Jong Hyun Park, Yongjun Park, Won Woo Ro and Murali Annavaram; (2016). "APRES: Improving Cache Efficiency by Exploiting Load Characteristics on GPUs".

[2] T. G. Rogers , M. O Connor and T. M. Aamodt; (2012) "Cache conscious wavefront scheduling"