GPU 캐시 이용율 초적화를 위한 동적 캐시라인 관리 방법

Maximizing GPU Cache Utilization with Adjustable Cache Line Management

ABSTRACT

Executing the irregular applications in general-purpose graphics processing units (GPGPUs) exposes serious challenges to their cache system. This paper proposes *JUSTIT*, an adjustable cache line management design that maximizes the GPU L1D cache utilization by being aware of the memory request access granularity. Specifically, *JUSTIT* can identify the 1-sector memory requests with a singular access and directly bypass L1D cache to prevent these memory requests from polluting the limited L1D cache space. For the other 1-sector memory requests, we redirect them to shared memory for future accesses. Our evaluation reveals that *JUSTIT* improves the IPC by 28%, compared to a state of the art memory management system.

1. INTRODUCTION

General-purpose graphics processing units (GPGPU) have become popular for many non-graphic applications such as big-data analytics and scientific programs, thanks to its massive computational parallelism and peak memory bandwidth.

To achieve high compute throughput, GPGPU adopts a coarse-grained (*CG*) memory hierarchy, in which its cache architecture and memory design are customized with larger cache line sizes and longer burst lengths, respectively. Additionally, the memory requests, belonging to a group of threads, called *warp*, are coalesced as larger

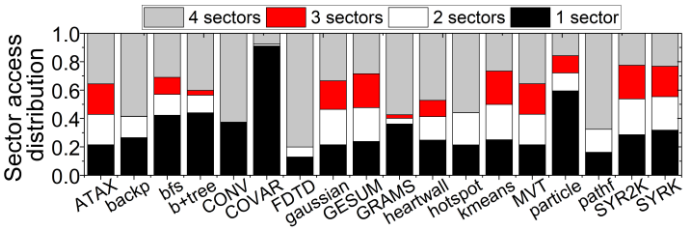

Figure 1: The number of sectors referenced in L1D cache blocks.

requests before accessing the memory hierarchy. While regular-structured programs can benefit from such design by exploiting their spatial locality and reducing the control overhead of memory requests, many emerging applications that exhibit irregular control flow and data access patterns, unfortunately fail to coalesce the memory requests and suffer from the inefficiency of the customized

cache and memory hierarchy [5]. Recent work, LAMAR [1], has focused on improving the memory bandwidth by modifying the underlying DRAM architecture to serve both the coalesced (coarse-grained) and un-coalesced (fine-grained/*FG*) memory requests. However, the FG memory requests expose serious practical challenges to the existing GPU cache design, which degrade the overall performance of GPGPU.

Placing FG memory accesses in GPU L1D cache can severely waste the cache space and make the cache under-utilized. We evaluate multiple workloads from different benchmarks [3,4] and summarize the number of 32B sectors in one 128B cache line that are referenced during their lifetime of L1D cache in Figure 1. As shown in the figure, applications with irregular memory access patterns, such as *FDTD*, always access 4-sector data blocks and can mostly utilize the whole L1D cache, while the irregular application, *COVAR,* which generates excessive un-coalesced memory requests, makes 91% of L1D cache lines under-utilized. In fact, 33% of L1D cache only accommodate 1-sector blocks across all workloads.

We propose an ad**JUST**able cache l**I**ne managemen**T** design (*JUSTIT*) to maximize the utilization of the GPU L1D cache by being aware of the memory access granularity. Prior work [2] also leverages such access

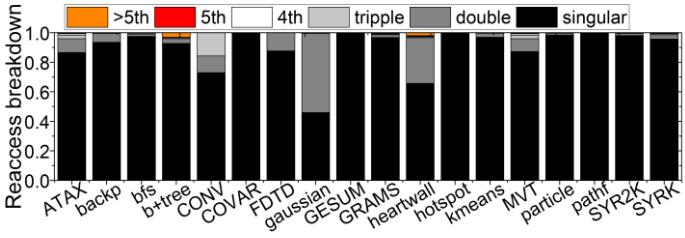

Figure 2: Data re-access time breakdown of 1-sector data blocks in L1D cache blocks.

granularity and compacts multiple 1-sector data blocks into single cache line. However, we observe that the proposed compaction technique is insufficient to fully utilize the L1D cache space. Figure 2 shows our key observation, which is the breakdown of different data re-access times in 1-sector data blocks during their lifetime in L1D cache. As shown in the figure, 89.7% of 1-sector data blocks only experience singular accesses. Since such data blocks exhibit no temporal locality and spatial locality, it is not beneficial to place them in L1D cache. To address this, we propose a *1-1 predictor* that identifies the memory requests, which access the 1-sector data blocks of singular (1) accesses. We then bypass these memory requests to avoid polluting the L1D cache space. On the other hand, for the 1-sector data blocks of multiple re-accesses, which only accounts for 2.1% of the total number of 1-sector data blocks, we propose to place these data blocks in the spare shared memory. We test the workloads from various benchmarks, and our evaluation reveals that 90.5% of shared memory is

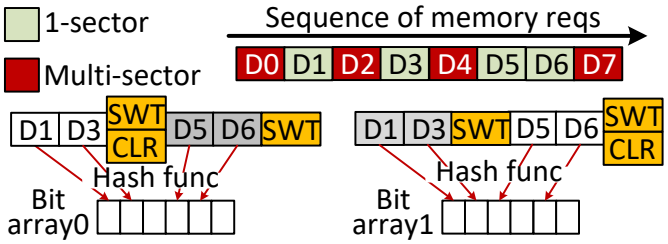

Figure 3: Overview of the proposed dual-bitarray counting bloom filter and the operations.

unused; the space is sufficient to accommodate all data.

2. MAIN DESIGN

1-1 predictor. We implement our predictor based on a counting bloom filter with a dual-bitarray to recognize the singular 1-sector data accesses and record the access history in the bitarray. Figure 3 shows an overview of our proposed dual-bitarray counting bloom filter, which consists of two *m*-counter bitarrays and *n*-hash functions. As shown in the figure, to record the access history, only the 1-sector memory requests are inserted in the bitarray by incrementing the counter values of the bitarray. We assume 1-sector memory requests are singular by default. To test if the 1-sector data blocks experience multiple accesses, the memory addresses are fed to the *n* hash functions, and we check if all the corresponding counters of the bitarray are higher than a threshold. To maintain the locality history, we also employ a dual-bitarray to periodically clear and swap, which can avoid making the mis-prediction due to the stale locality data.

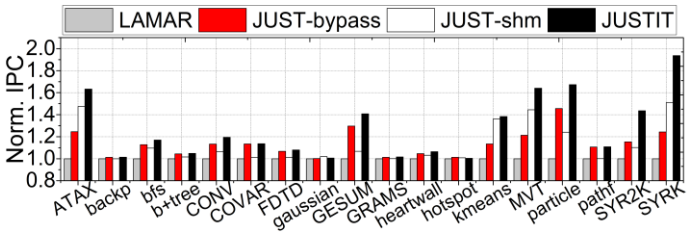

Figure 4: Overall GPU performance in terms of IPC.

3. PERFORMANCE IMPROVEMENT

We configure four different GPU systems, which are *LAMAR*, *LAMAR* with bypass (*JUST-bypass*), *LAMAR* with SHM (*JUST-shm*) and *LAMAR* with both bypass and SHM (*JUSTIT*). Figure 4 shows that *JUSTIT* improves the IPC performance by 28%, compared to *LAMAR*.

4. CONCLUSION

This paper proposes *JUSTIT* to maximize the GPU L1D cache by being aware of the memory access granularity. The performance of *JUSTIT* is 28% better than *LAMAR*.

REFERENCE

[1] Minsoo Rhu *et al.*, "A locality-aware memory hierarchy for energy-efficient GPU architectures." In MICRO (2013).

[2] Bingchao Li *et al.*, "Elastic-cache: GPU cache architecture for efficient fine-and coarse-grained cache-line management.".

[3] S. Chen *et al.*, "Rodinia: A benchmark suite for heterogeneous computing." in IISWC (2009).

[4] Pouchet *et al.*, "Polybench: The polyhedral benchmark suite."

[5] G. Koo *et al.*, "Access Pattern-Aware Cache Management for Improving Data utilization in GPU." In CAL (2017).