Memory Coalescing for Hybrid Memory Cube

Xi Wang
Texas Tech University
xi.wang@ttu.edu

John D. Leidel
Texas Tech University
john.leidel@ttu.edu

Yong Chen
Texas Tech University
yong.chen@ttu.edu

ABSTRACT

Arguably, many data-intensive applications pose significant challenges to conventional architectures and memory systems, especially when applications exhibit non-contiguous, irregular, and small memory access patterns. The performance of data-intensive workloads running on traditional processors and architectures is not as expected, especially when encountering the irregular memory-access patterns that produce random memory footprints. The long memory access latency can dramatically slow down the overall performance of applications. In addition, the growing data-level parallelism in these data-intensive workloads increase the concurrency in data accesses and further complicate the memory system support.

In order to reduce the latency of memory accesses, two main research directions exist in current studies. The first direction focuses on moving computation to data, thus reducing the need and the memory traffic of moving data between memory and processors. These efforts include near data processing (NDP) and processing in memory (PIM)[1, 2, 3, 4, 5]. While these techniques bridge the gap between the processors and main memory, they cannot eliminate memory accesses, thus the bandwidth of conventional DDR devices restricts the ceiling of the overall performance.

The second direction focuses on developing advanced memory devices that satisfy the growing desire of high memory bandwidth and low latency access. This motivation stimulates the advent of novel 3D-staked memory devices such as the Hybrid Memory Cube (HMC), which provides significantly higher bandwidth compared with the conventional JEDEC DDR devices [6, 7].

Even though many existing studies have been devoted to achieving high bandwidth throughput of HMC, the bandwidth potential cannot be fully exploited due to the lack of highly efficient interfacing methodology designed and optimized for HMC devices. As the size of the memory request transactions is consistent with the cache line size and the fixed burst size in DDR interface, the memory request sizes are usually fixed as 64B in mainstream architectures. It may evoke higher latency and control overhead to apply the same policy to the flexible packet-based HMC devices naively.

In this research, we introduce a novel memory coalescer methodology that facilitates memory bandwidth efficiency and the overall performance through an efficient and scalable memory request coalescing interface for HMC. The memory coalescer is positioned between the last level data cache and the MSHRs. We overlap the latency of the memory coalescing with the cache blocking time when there is no available MHSR entry to hold more cache misses. Three major components are introduced in the memory coalescer: *pipelined request sorting network*, *DMC unit* and *dynamic MSHRs*.

The pipelined request sorting network is responsible for sorting all memory requests (misses from LLC), including the load/store miss and write back requests. Once the sorting completes, the ordered requests are flushed to the DMC unit.

The DMC unit is responsible for the first-phase memory coalescing, which constructs large request packets. When the sorting procedure completes, the DMC unit takes over the sorted requests and coalesces the adjacent memory accesses into larger request packets. As soon as the coalescing is accomplished, the combined requests are immediately pushed into the coalesced request queue (CRQ).

A conventional MSHR entry usually consists of three segments: requested block address, the hardware-dependent target information, and a valid bit that indicates whether an MSHR entry is in use. As the conventional MSHR entry only holds the misses to a single cache line, the size of the misses is always the same as the cache line size. However, the size of coalesced requests could be either one or multiple cache lines. Therefore, a 2-bit segment named *size*, is appended to each MSHR entry that represents the coalesced request size.

ICPP 2018 August 13 – 16, Eugene, Oregon

DOI: 10.475/123_4

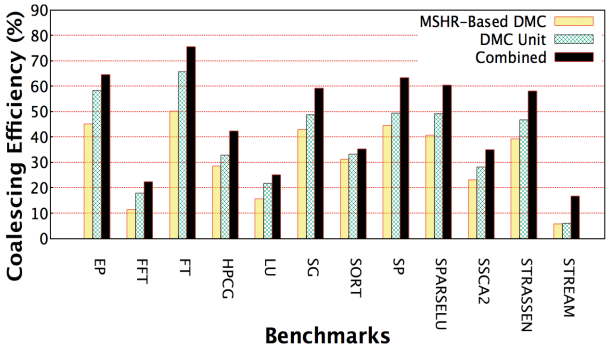

Figure 1: Coalescing Efficiency of the Memory Coalescer

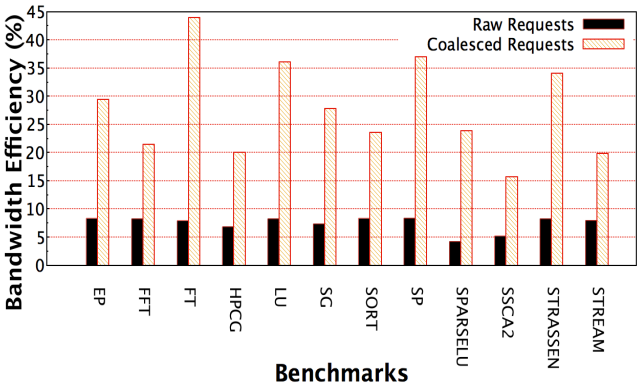

Figure 2: Bandwidth Efficiency of Coalesced and Raw Requests

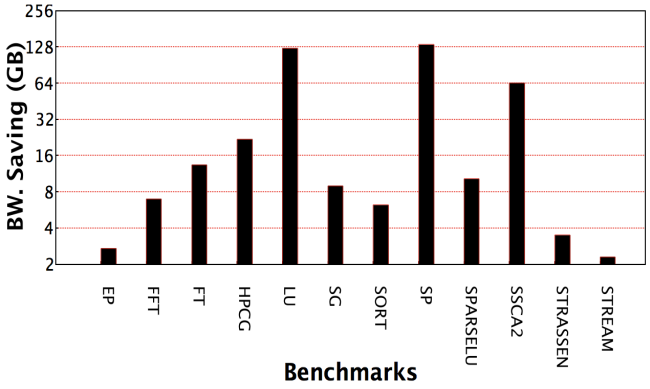

Figure 3: Bandwidth Saving

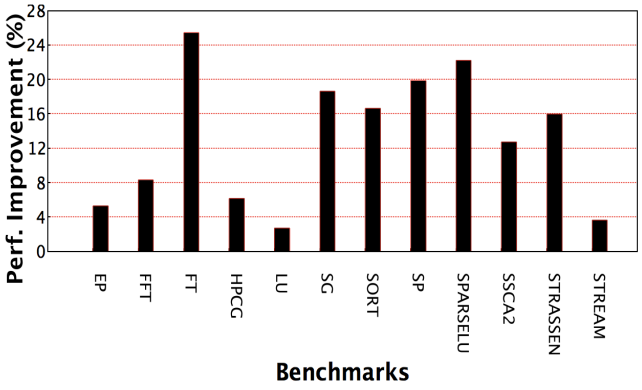

Figure 4: Performance Improvement with Memory Coalescer

We present the design and implementation of this approach on RISC-V embedded cores with attached HMC devices. Our evaluation results show that the new memory coalescer eliminates 47.47% memory accesses to HMC, as shown in the Figure 1.

Besides the coalescing efficiency, we also investigate the bandwidth utilizations. As shown in the Figure 2, the average bandwidth efficiency of the raw requests is only 7.43%. In contrast, conveying the coalesced requests attains a bandwidth efficiency of 27.73%, which is approximately 4X improvement on the bandwidth utilization exhibited by the memory coalescer. In order to quantify the memory coalescer's impact on bandwidth, the total bandwidth savings are also measured. As reported in the Figure 3, the memory coalescer saves, on average, 33.25GB of unnecessary control data transfer across all test cases.

Finally, we record and analyze the runtime statistics in order to study actual application speedup using our approach. As reported in the Figure 4, the majority of test cases demonstrated the runtime improvement by over 10%. Particularly, the performance improvement of FT and SparseLU achieved 25.43% and 22.21%, respectively. On average, the memory coalescer delivered 13.14% performance gain across all test suites. These results indicate that our approach has a positive impact on the performance of data-intensive applications.

1. REFERENCES

[1] Mingyu Gao et al. Practical near-data processing for in-memory analytics frameworks. In *PACT 2015*.

[2] Kevin Hsieh et al. Transparent offloading and mapping (tom): Enabling programmer-transparent near-data processing in gpu systems. In *ISCA 2016*.

[3] Ping Chi et al. Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory. In *ISCA 2016*.

[4] Junwhan Ahn et al. Pim-enabled instructions: A low-overhead, locality-aware processing-in-memory architecture. In *ISCA 2015*.

[5] Dongping Zhang et al. Top-pim: throughput-oriented programmable processing in memory. In *HPDC 2014*.

[6] HMC Specification 2.1. Technical report, December 2015.

[7] Shaizeen Aga and Satish Narayanasamy. Invisimem: Smart memory defenses for memory bus side channel. In *ISCA 2017*.