Memory Hotspot Optimization for Data-Intensive Applications

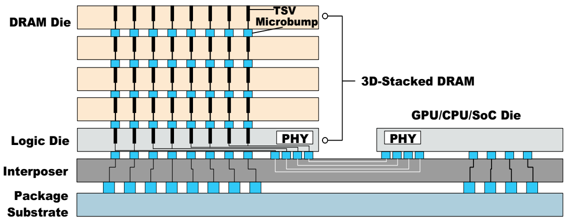

Fig. 1: Example of 3D-Stacked Memory Layout

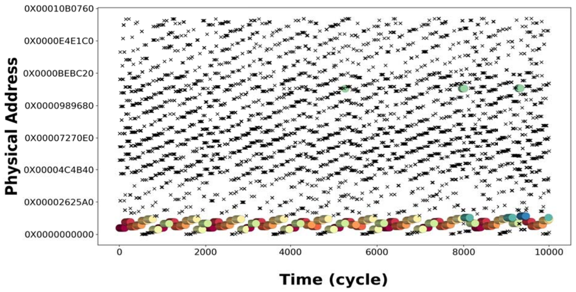

Fig. 2: SSSP

I. Extended Abstract

Emerging High-Performance Computing (HPC) workloads, such as graph analytics, machine learning, big data science, are data-intensive. The data-intensive workloads usually present irregular memory footprints with limited data locality, and thus incur frequent cache misses and a growing desire for memory bandwidth. Driven by this need, 3D-stacked memory devices such as Hybrid Memory Cube (HMC) and High Bandwidth Memory (HBM) are introduced to yield significantly higher throughput. However, the traditional interfaces and optimization methods for JEDEC DDR devices cannot fully exploit the potential performance of 3D-stacked memory to handle massive irregular memory accesses accompanied with data-intensive applications.

3D-stacked memory devices (as shown in Figure 1), such as the High Bandwidth Memory (HBM) [1] and Hybrid Memory Cube (HMC) [2], provide significantly higher bandwidth with respect to conventional Double Data Rate synchronous Dynamic Random Access Memory (DDR DRAM), and offer an opportunity to better address requirements of data-intensive applications. In these devices, the DRAM dies are stacked on top of a logic die via 3D packaging. The logic layer implements the memory controller that manages the stacked DRAMs. Well known commercial devices using this technology are the latest generations of NVIDIA's Graphic Processing Units (GPUs), Intel's Xeon Phi processors and Fujitsu PrimeHPC FX100.

One issue for data-intensive applications are the frequent generation of memory hotspots, due to the fine-grained nature of their data accesses. Memory hotspots are frequently accessed memory locations that may significantly hinder the performance of DRAM devices, due to their banked design. In fact, frequent accesses to the same memory banks lead to increased bank conflicts of the memory operations, thus lengthening their latency [3]. Given nondeterministic memory footprints presented in the irregular applications, the bank-interleaving may not be able to avoid the hotspot formations as expected.

We cluster the memory traces depending on the values of physical addresses to identify any frequently accessed memory regions. Since we cannot predict the number of clusters, we use the unsupervised density-based spatial clustering of applications with noise (DBSCAN) algorithm [4]. We set the epsilon distance of DBSCAN to 1KB, which is equivalent to the row size of HBM, to group adjacent memory accesses. We set the time windows for the analysis to 10,000 clock cycles.

As shown in Figure 2, circles represent request clusters, and distinct clusters are differentiated by colors. Crosses identify unclustered requests with limited data locality. We observe multiple request clusters, demonstrating the presence of memory hotspots. Consequently, some DRAM banks are more frequently accessed than others. Higher bank utilization leads to a higher probability of bank conflicts [3]. However, if we were able to monitor memory hotspots, we could employ the information to optimize the performance of the memory itself when executing data-intensive applications.

As such, we propose a novel Hotspot-Aware Manager (HAM) infrastructure for 3D-stacked memory devices that is capable of optimizing memory access streams via request aggregation, hotspot detection, and in-memory prefetching. HAM is a generalized design that is applicable to different 3D-stacked memory devices, such as HBM and HMC, as shown in Figure 3. For HBM that consists of multiple independent channels with 2 HBM channels per DRAM die [1], we host a HAM unit inside of each HBM channel controller to manage the local accesses, as demonstrated in Figure 3. In the case of HMC, the stacked DRAMs are vertically partitioned into 32 vaults and every 8 vaults are grouped as a quadrant [2]. As each vault within a quadrant shares the same HMC link to transfer data between HMC and processors, the quadrant-based HAM is employed rather than a vault-based design

978-1-7281-3613-4/19/$31.00 ©2019 IEEE
DOI 10.1109/PACT.2019.00048

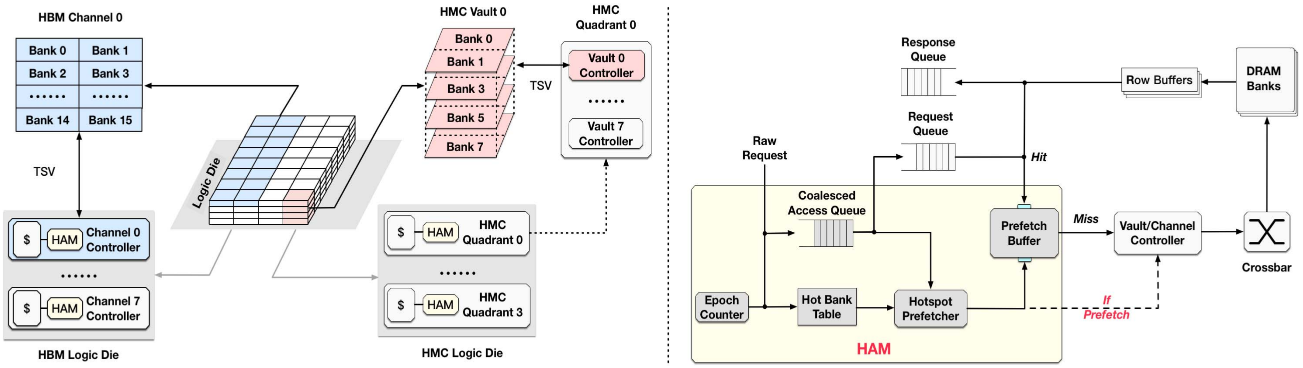

Fig. 3: Architecture of HAM in 3D Stacked Memory

to minimize the space overhead and die area occupation, as shown in Figure 3.

As illustrated on the right side of Figure 3, HAM consists of four major components: *Coalesced Access Queue*, *Hot Bank Table*, *Hotspot Prefetcher*, and *Prefetch Buffer*.

The coalesced access queue (CAQ) is a First In First Out (FIFO) queue that aggregates raw requests from processors based on the respective row addresses. Each CAQ entry merges the requests hitting the same DRAM row and signals the prefetcher when poping out the requests from CAQ.

The hot bank table (HBT) records the number of accesses to each DRAM bank in the 3D-stacked memory and indicates whether a requested bank is hot or cold.

The hotspot prefetcher manages the prefetching logic. As soon as a request is received from CAQ, prefetcher will check whether requested row is cached or not. If the row is found in the prefetcher buffer, then no prefetching is needed. Otherwise, the prefetcher continues checking the target row and bank status of this request. Once a hot row or hot bank is recognized, then a prefetching request is issued to the specific vault or channel and prefetch the entire row.

We also employ a two-port prefetch buffer in the HAM design. The first port handles the access streams from the request queue. While the second port tackles inquiries from the hotspot prefetcher, which only inquires the addresses of prefetch buffer entries. This implies that requests from prefetcher never manipulate the data cached in the prefetch buffer. In this circumstance, hardware hashing functions [5] are utilized for referencing the prefetch buffer to reduce space and energy overhead induced by the replicated hardware comparators for both ports.

We evaluated HAM by simulating it in an architecture based on RISC-V cores (implementing RV64IMAFDC ISA) [6] with an attached HMC device. To compare the performance, we implemented 3 more prefetching designs besides the HAM. The baseline scheme (BASE) implements a memory-side streaming prefetcher that loads the entire row (256B) to the prefetch buffer if a miss occurs. The second case only implements the CAQ for prefetching, which prefetches the row if it is requested by two or more read requests in the CAQ. Accordingly, the third case relies on the HBT that only prefetches the data residing in the hot banks. Evidently, HAM shows highest hit rate (61.21%) for the prefetch buffer, which achieves an average of 4.19X improvement compared with the baseline case. Besides, on average, the hit rates of CAQ and HBT are improved by 2.41X and 3.40X, respectively. As such, HAM reduces a great amount of redundant memory accesses to the 3D-stacked memory and boost the overall performance of the memory system for data-intensive workloads.

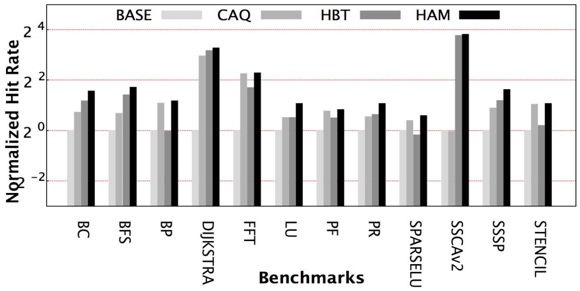

Fig. 4: Prefetch Buffer Hit Rate

References

[1] " JEDEC Standard High Bandwidth Memory(HBM) DRAM Specification," https://www.jedec.org/standards-documents/docs/jesd235a, Tech. Rep., 2013.

[2] "Hybrid Memory Cube Specification 2.1," Tech. Rep., December 2015. [Online]. Available: http://www.hybridmemorycube.org/files/SiteDownloads/HMC-30G-VSR_HMCC_Specification_Rev2.1_20151105.pdf

[3] D. Kaseridis, J. Stuecheli, and L. K. John, "Minimalist open-page: A dram page-mode scheduling policy for the many-core era," in *2011 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2011, pp. 24–35.

[4] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *Kdd*, vol. 96, no. 34, 1996, pp. 226–231.

[5] M. Ramakrishna, E. Fu, and E. Bahcekapili, "Efficient hardware hashing functions for high performance computers," *IEEE Transactions on Computers*, vol. 46, no. 12, pp. 1378–1381, 1997.

[6] A. Waterman, Y. Lee, R. Avizienis, D. A. Patterson, and K. Asanovi, "The RISC-V Instruction Set Manual Volume II: Privileged Architecture Version 1.7," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2015-49, May 2015. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-49.html