ReTagger: An Efficient Controller for DRAM Cache Architectures

Mahdi Nazm Bojnordi
University of Utah
bojnordi@cs.utah.edu

Farhan Nasrullah
University of Utah
farhan.nasrullah@utah.edu

ABSTRACT

3D die-stacking has enabled energy-efficient solutions for near data processing by integrating multiple dice of high-density memory layers and processor cores within the same package. One promising approach is to employ the in-package memory as a gigascale last-level cache for data-intensive computing. Most existing in-package cache controllers rely on the command scheduling policies borrowed from the off-chip DRAM system. Regrettably, these control policies are not specifically tailored for in-package cache traffics, which results in a limited bandwidth efficiency. This paper proposes ReTagger, a DRAM cache controller that employs repeated tags to alleviate the cost of DRAM row buffer misses. Our simulation results on a set of ten data-intensive applications indicate an average of 20% performance improvement for the proposed controller over the state-of-the-art DRAM caches.

ACM Reference Format:
Mahdi Nazm Bojnordi and Farhan Nasrullah. 2019. ReTagger: An Efficient Controller for DRAM Cache Architectures. In *The 56th Annual Design Automation Conference 2019 (DAC '19), June 2–6, 2019, Las Vegas, NV, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3316781.3317895

1 INTRODUCTION

As the demand for *big data* processing increases [1, 2], memory bandwidth wall [3] and data movement problems [4] escalate in all forms of computing systems from datacenters to mobile applications. One key response from industry to this ever-growing problem has been the integration of disparate technologies within the same package using 3D die-stacking to reduce the amount of off-chip data movement. For example, Micron's hybrid memory cube (HMC) stacks multiple DRAM layers on a flexible logic layer that communicate through energy-efficient and fast through silicon vias (TSVs) [5, 6]; Intel integrates up to 16 GB of memory in a multi-channel DRAM (MCDRAM) with 4× higher bandwidth than DDR4 in Knights Landing processors [7]. As compared with off-chip memory systems, in-package integration provides up to 10× more bandwidth with a significantly lower power and a smaller footprint, which make it an attractive solution for accelerating a variety of data intensive applications from scientific and engineering domains [8–10]. One promising approach that has been examined by researchers in academia and industry is to employ the in-package memory as a gigascale last-level cache [11–13]. However, due to the significant inefficiencies of the in-package memory interface for cache traffics, it is unclear if the existing solutions could address all the requirements of user applications in future commodity computers.

2 BACKGROUND

Figure 1 shows an example processor (or accelerator) system that consists of an in-package memory and multiple processor cores. The processor cores and the in-package memory layers communicate through a high bandwidth interface that includes high-bandwidth controllers, TSVs, and often microbumps within a silicon interposer [14]. The memory layers are typically divided into 4-8 independent channels (a.k.a., vaults) that provide the highest level of parallelism in the memory system. Each channels is further divided into 8–16 banks, which results in a total of 128 banks to exploit data-level parallelism in user applications. The DRAM banks are organized as arrays of rows×columns, sharing common data and address TSVs and on-die buses. Serving every read or write request requires sending an appropriate sequence of commands—i.e., `precharge`, `activate`, `read`, and `write`—to the DRAM dice. Moreover, a set of timing and power constraints defined by the interfacing standard, e.g., JEDEC WideIO [15], dictate the minimum delay between each pair of commands issued to the memory system.

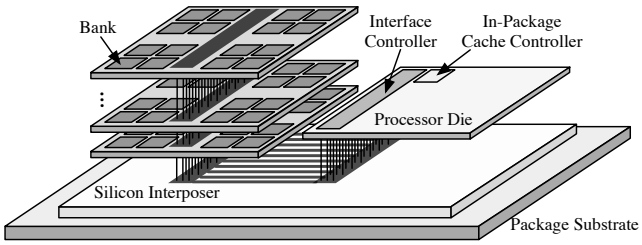

Figure 1: Illustrative example of a high bandwidth memory (HBM) system.

Most existing proposals for in-package cache architectures borrow the control policies from off-chip memory controllers to perform command scheduling, refresh management, quality of service maintenance, and power optimization. However, major differences between in-package and off-chip memory interfaces, as well as the inherently different cache traffic from main memory result in unprecedented power and performance challenges that may not be effectively addressed through the existing control policies.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
DAC '19, June 2–6, 2019, Las Vegas, NV, USA
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6725-7/19/06…$15.00
https://doi.org/10.1145/3316781.3317895

2.1 DRAM Controllers

A DRAM controller receives a request stream from the processor, and generates a corresponding DRAM command stream. Every memory request needs to access a block of data within a row of the DRAM subsystem. An `activate` command is required to load the row into (local and global) row buffers prior to accessing data. Consecutive accesses to the same row, called *row hits*, enjoy the lowest access energy and latency. A *row miss* may occur if the row buffers do not contain the desired row, which may require a `precharge` command that precharges the bitlines prior to activating the next row. Over the past two decades, DRAM control policies have been extensively studied in the context of off-chip main memory systems [16–26]. Other proposals aimed to reduce the adverse effects of DRAM refresh on system performance [27, 28].

2.2 Existing In-Package Cache Proposals

The architecture of large DRAM cache has achieved significant attention in recent years mainly to alleviate the key problems of cache design. There exists two main categories of DRAM cache architectures based on storage, allocation, and replacement policies for tag and data: *fine granularity* and *coarse granularity* DRAM cache. Fine granularity DRAM cache mainly manages 64byte cache lines and requires significant storage for the tags, which makes it impractical to keep the tags in on-chip SRAM. For example, a 1GB DRAM cache needs 128MB tag storage, where each tag is 8 bytes wide. For the same reason, almost all state of the art DRAM cache designs propose to store tags in the in-package DRAM, thereby introducing tag access latency and bandwidth consumption. Recent proposals on Alloy [29] and Loh-Hill [30] cache architectures have examined techniques to improve access latency. Alloy cache [29] is a direct map cache whose tag storage, cache allocation, and replacement mechanism work on cache line granularity. Data and tag are placed adjacent to each other and accessed together, which helps to improve cache read hit latency. However, read misses or write accesses do not get benefit from this and cause extra traffic on both in-package DRAM channel and off-chip DRAM.

The Bear cache [31] proposes three optimization techniques to reduce the unnecessary traffic on DRAM channel for cache access. Bandwidth aware bypass scheme predicts cache data reuse behavior in run time and bypasses cache miss-fill during low re-use situation. The bandwidth efficient write probe and on-chip neighboring tag cache (NTC) helps to optimize DRAM cache bandwidth by removing some of the tag check accesses to DRAM. R-Cache [32] proposes an RRAM based in-package memory that eliminates the bandwidth overhead of tag checks via in-situ comparison.

Unison cache [33] is a page granularity set associative DRAM cache which improves cache hit latency by reading all the tags and a predicted data block. If the hit prediction is correct, the access is roughly the same as a single DRAM access; otherwise, it is doubled. Coarse granularity DRAM cache pays huge bandwidth penalty on every cache miss due to loading the entire page even if the program does not exhibit significant spatial locality. In its worst case, if the victim block is dirty, a write back for the entire page will be issued to main memory. This consumes significant bandwidth of off-chip DRAM channel which is already limited. TDC [34] proposes a new software-hardware based architecture which obviates the tag checking operation for page granularity DRAM cache. It, however, adds software complexity and extra on-chip buffer to maintain the address mapping coherency. FTDC [35] and Banshee [36] adopt similar architecture to TDC [34] but also improve off-chip memory bandwidth by introducing frequency based reuse prediction. The mechanism avoids cache replacement on every miss; it performs a cache replacement only if the currently accessed block is predicted to be highly reused in future based on reuse counter.

3 DESIGN OVERVIEW

A significant challenge in designing gigascale cache architectures is often the limited bandwidth efficiency of memory. We propose ReTagger that optimizes the tag and data management policy of the in-package cache with respect to the DRAM row buffer status.

3.1 Tag and Data Management in DRAM Cache

A typical coarse-grained cache may provide 1–4KB cache blocks, hence it may significantly suffer from moving unnecessary data between cache and main memory on every tag miss and block eviction [33–36]. Fine-grained cache architectures employ smaller cache blocks (e.g., 64B) to reduce the amount of unnecessary data movement between main memory and in-package cache. The fine block granularity, however, results in significant memory and bandwidth overheads for tag management. A fine-grained DRAM cache needs to manage a large amount of meta data (including dirty, valid, tag, and error correction bits) per each cache block [29–31]. To alleviate the memory costs, meta data is typically collocated with the data blocks in the DRAM layers. For example, a cache block of 64B data may need 8B meta data, which translates to an additional 12.5% DRAM storage. Moreover, tag checks may exert a substantial bandwidth overhead to the in-package memory interface and may increase the cache access latency. As a result, tag and data management has become the main concern in DRAM cache architectures.

Figure 2 illustrates general forms of the existing proposals for direct mapped and set associative in-package cache systems. In the direct mapped approach, a data block and its tag are accessed using a single read; therefore, the data can be immediately used on a read hit [29, 30]. An additional write may be required to update the cache block on a write hit. All tag mismatches result in forwarding the cache requests to main memory. However, installing the cache block in the DRAM cache is up to the cache controller. For example, the Bear cache employs stochastic mechanisms to skip the DRAM cache and forward the requests to main memory [35].

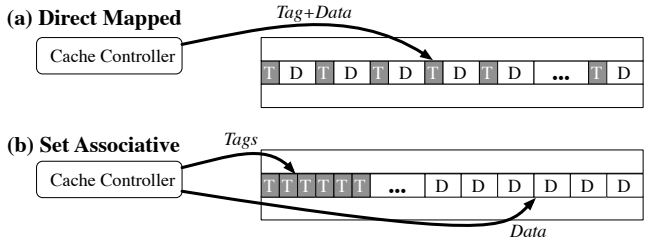

Figure 2: General forms of the existing DRAM cache proposals.

Figure 2 (b) shows another form of caching that reads multiple cache tags first and reaches the cache data in the second access. This way, both read and write accesses need two steps on hit. Similar to the direct mapped mechanism, a tag mismatch is detected after the first access and will be forwarded to the main memory. This two-step approach is suitable for building set associative DRAM caches [30]. However, recent work has shown a limited performance beyond 2-way set associativity for DRAM caches [37].

3.2 Row Buffer Access

The existing DRAM cache proposals mainly focus on improving bandwidth and cache access latency through novel tag layouts, way prediction, block installation, and cache access suppression. This paper considers an often overlooked detail in performing DRAM operations to further optimize the performance and energy-efficiency of DRAM cache controllers. Prior to any tag checks for an incoming request, the cache controller needs to ensure that the target DRAM row is placed in the row buffer. Satisfying this requirement, the controller may need to issue `precharge` and `activate` commands to the DRAM layers. Only then, a read request can be issued to access the Tag or Data. For an incoming cache request, a *row hit* occurs if the target row is already placed in the row buffer; otherwise, the request encounters a *row miss*. Therefore, each incoming cache request encounters one of the four possible conditions based on the row buffer status and the outcome of tag comparison. Figure 3 shows the four possibilities for two example applications, namely FT and HIST. A relatively larger fraction of the cache requests in FT benefit from row hits, which are faster and more energy efficient than row misses. However, the majority of cache requests for both FT (71%) and HIST (90%) suffer from the high energy, bandwidth, and latency overheads of row misses. The worst case is a row miss resulting in a tag mismatch, thereby necessitating a main memory access after a row miss.

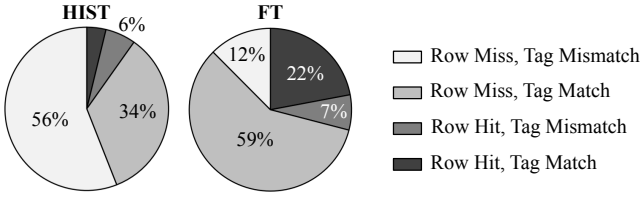

Figure 3: Possible tag check scenarios with respect to the row buffer status.

3.3 The Proposed Row Scheduler

The goal of this paper is to reduce the diverse impacts of row buffer misses on the cache performance and energy efficiency. We propose a *row scheduler* that can be integrated in the existing cache controllers for managing row buffers and expediting the tag check process. Figure 4 illustrates an overview of the proposed DRAM caching mechanism using the row scheduler. Similar to the existing fine-grained DRAM caches, the proposed mechanism relies on partitioning rows into blocks of data and tags. For example, a 2KB row buffer can store up to 28 blocks, each of which comprises 64B data and 8B metadata. (A similar partitioning approach is employed by the prior work on Bear [31].) Please notice that 32B of each row remains unused. The proposed row scheduler makes use of these otherwise unused bytes to repeat the tag bits of all the blocks within each row. Assuming that every block has an eight-bit tag, a total of 28B is needed to store the repeated tags (ReTag). ReTags are store as the last bytes of the row so that they can be read or written in one access.

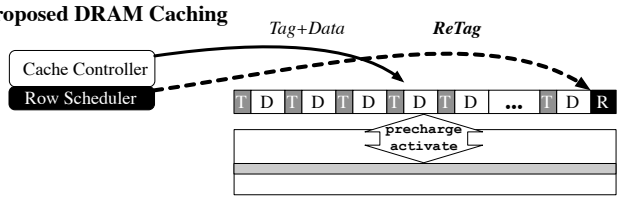

Figure 4: Illustrative example of the proposed DRAM caching mechanism.

Key Idea. The key idea of the proposed row scheduler is to monitor the cache traffic at run time and proactively prepare the row buffers for future incoming cache requests. To reduce the impacts of row misses, the row scheduler sends a `precharge` command to a bank if there is no outstanding requests on that bank at the cache controller. This is similar to the basic closed row (page) policy employed by off-chip DRAM controllers. However, closing the row buffer can impact the future row hits and misses both: (1) a row miss may be serviced faster if the row buffer is empty and (2) a row hit may become a row miss after issuing the `precharge` command. We observe that a closed row policy is not beneficial for the evaluated applications mainly because of the latter issue (see Section 5).

Instead, the row scheduler employs ReTags to reduce the cost of converting possible row hits to misses; while, it opportunistically closes the row buffer as soon as all of its outstanding requests at the cache controller are serviced. The proposed scheduling policy is orthogonal to the existing cache control policies and can be used to further improve the bandwidth efficiency and performance of in-package caches. The ReTag bits enable the controller to make a performance critical decision on if a request should be serviced at the cache or to be forwarded to the main memory. This decision can only be made if the cache controller knows the outcome of tag comparison, which may be significantly delayed due to a row buffer miss. The proposed architecture fetches the row ReTags and stores them on the processor die before closing a row buffer. Thus, the controller can employ the ReTags to determine if a future cache request leads to a match or mismatch. The results of such pre-processing at the cache controller can improve the access latency and cache bandwidth through

- accelerating the decision made for those requests with "row miss, tag mismatch" in Figure 3 and forwarding them to the main memory; and
- eliminating the need for tag check reads in the case of write requests that result in "row miss, tag match" and making them single HBM access rather than double.

The main challenge, however, is the overhead of accessing ReTags per each row. To reduce the bandwidth and storage costs, (1) ReTags are designed to contain only the tag bits rather than all of the metadata, (2) a copy of all the tags in a DRAM row are bundled and accessed in one shot, (3) each row stores its own ReTags therefore

the tags are accessed quickly when the row is open, and (4) the ReTags are only accessed if no outstanding requests for the bank exists.

4 PROPOSED ARCHITECTURE

This section provides a system overview and explains the proposed architectural mechanism for row scheduling.

4.1 System Overview

Figure 5 shows an overview of the proposed architecture used by a multicore system with an on-chip cache connected to the HBM layers via an eight-channel WideIO interface. An off-chip DRAM system is employed as the main memory under a DDR4 interface. Similar to prior work on fine-grained caches [29, 31], tags are collocated with the data blocks in the HBM layers. Moreover, each HBM channel is provided with an HBM controlling unit that comprises a *cache controller* for block management and a *command scheduler* for generating WideIO commands—e.g., `refresh`, `precharge`, `read`, and `write`. The cache controller is designed after the state-of-the-art DRAM cache controllers [29, 31] and is responsible for high level block management tasks, such as receiving requests form the processor side, accessing tags in the HBM layers, performing tag checks to determine match/mismatch, installing a cache block in the HBM cache on a miss, evicting a cache block from the HBM and producing writeback traffic to the main memory if the evicted blocks are dirty, monitoring the cache bandwidth and skipping the HBM if necessary to improve performance.

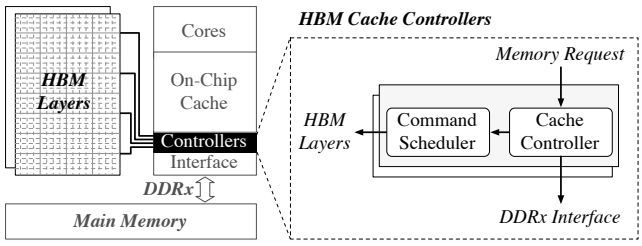

Figure 5: Illustrative top view of the proposed ReTagger architecture in a multicore system.

We observe that command scheduling plays a significant role in improving the efficiency of HBM cache system. Figure 6 shows the relative execution times for a set of ten parallel applications using three different scheduling policies—namely FCFS, FR-FCFS (O), and FR-FCFS (C)—based on the algorithms from prior work on DRAM access scheduling [16].[1] The FCFS policy implements a first-come first-serve algorithm for servicing the requests in order; whereas, the FR-FCFS policy implements an out-of-order algorithm for prioritizing the ready row hits over long latency row miss requests. The out-of-order scheduling exhibits a superior performance compared to the FCFS algorithm. In this experiment, we consider two variations of the FR-FCFS algorithm with respect to row buffer management. FR-FCFS(O) represents an open row policy where the row buffer holds the currently open row even after serving the last outstanding request. This policy will be beneficial to the future accesses made to the same row. FR-FCFS(C) is the closed row version of the algorithm that sends a `precharge` to any active banks that has no outstanding requests at the scheduler. This policy is well-suited for any future request that would be a miss in the row buffer. We observe that the two row management policies perform similarly for the HBM cache system.

[1]More details on the experimental setup are provided in Section 5.

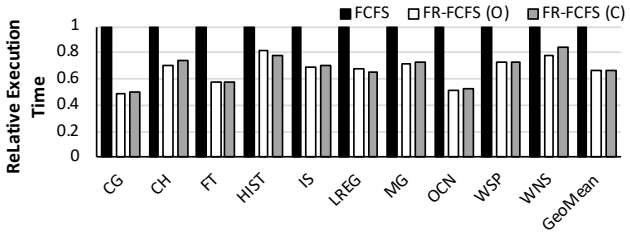

Figure 6: Performance of various policies for HBM cache scheduling.

4.2 Proposed ReTag Management

Similar to the closed row policy, the proposed row scheduler sends a `precharge` command to any bank that has no outstanding requests at the cache controller. Figure 7 shows the proposed mechanism for managing ReTags at the row scheduler. Prior to generating any WideIO commands, all incoming cache requests are inserted into a *scheduling queue*. The command scheduler will then access the queue to issue proper WideIO commands. The ReTag Manager supplements the command scheduler by monitoring the scheduling queue and issuing commands to the HBM layers. Two ReTag management tasks are necessary at the row scheduler.

- **ReTag Fetch** is an operation required to read the ReTags from HBM layers before closing the current row. This operation is performed on a row buffer using a single WideIO read. A *no outstanding* signal is used to indicate when the ReTag manager can issue the read command.
- **ReTag update** is only required when a new block is installed in the HBM cache. The ReTag manager keeps track of all the installed blocks from the time a row is opened in the row buffer. Right before closing the row, a single write may be issued to update the ReTags only if at least one block install has been performed to the row.

The proposed controller requires an insignificant amount of additional hardware for generating the *no outstanding* signal and storing/tracking the ReTags. For an HBM cache channel with a total of 16 banks using 2KB rows to store 64B data blocks, the on-die storage overhead is only 448 bytes.

ReTag Management in the Row Scheduler

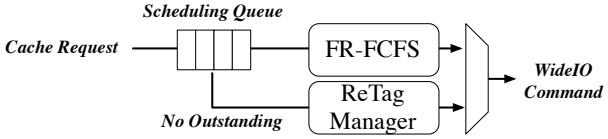

Figure 7: ReTag management in the row schedule.

5 EVALUATIONS

This section explains the evaluation methodologies and results for the proposed DRAM caching mechanism.

5.1 Methodology

We evaluate the power/energy and delay of the proposed architecture based on hardware synthesis with the FreePDK [38] library at the 45nm CMOS technology. We use McPAT [39] to estimate the overall processor power consumption. To assess the energy and performance potentials of DRAM memories, CACTI IO [40], Micron power calculator [41, 42], and DRAMPower [43] are used. We use a similar approach to the one used in the prior work [44] to estimate the energy consumption of the HBM memory system. A heavily modified version of ESESC [45] is used to model a multicore processor using the proposed 3D DRAM cache system. For the baseline DRAM cache architecture, we implement the state-of-the-art controller proposed by the Bear cache [31]. Table 1 shows the simulation parameters for the evaluated systems.

Table 1: System parameters.

Core	four 4-issue OoO cores, 128 ROB entries, 3.2 GHz
IL1/DL1 cache	32KB, 4-way, LRU, 64B block, hit/miss delay 1/1
L2 cache (shared)	4MB, 8-way, LRU, 64B block, hit/miss delay 8/2, MESI protocol
Temperature	360 K (77 °C)
HBM 1GB	8 channels, 1 rank/channel, 16 banks/rank, DDR4, 800MHz, tRCD: 44, tCAS: 44, tRP: 44, tRTP: 46, tRAS: 112, tWR: 4
DDR4 32GB	1 channel, 2 ranks/channel, 8 banks/rank, tRCD: 44, tCCD: 61, tWTR: 31, tWR: 4, tRTP: 46, tRP: 44, tRRD: 16, tRAS: 112, tRC: 271, tFAW: 181

A mix of ten data intensive parallel applications from Phoenix [46], NAS [47], and SPLASH-2 [48] benchmark suites are used to evaluate the performance potentials of the proposed caching mechanism. We run the simulations until completion for power and performance evaluations. Table 2 summarizes the evaluated benchmarks and their input sets.

Table 2: Applications and data sets.

Label	Benchmarks	Suite	Input
FT	Fourier Transform	NAS OpenMP	Class A
IS	Integer Sort	NAS OpenMP	Class A
MG	Multi-Grid	NAS OpenMP	Class A
CG	Conjugate Gradient	NAS OpenMP	Class A
CH	Cholesky	SPLASH-2	tk 15.0
OCN	Ocean	SPLASH-2	514x514 ocean
WSP	Water-Spatial	SPLASH-2	512 molecules
WNS	Water-NSquared	SPLASH-2	512 molecules
HIST	Histogram	Phoenix	100MB file
LREG	Linear Regression	Phoenix	50MB key file

5.2 Simulation Results

Performance. Figure 8 shows the relative execution times of the evaluated applications normalized to the FR-FCFS(O) baseline. The results indicate that the proposed ReTag mechanism can reduce the average execution time by 20% compared to the FR-FCFS(O) baseline. The FR-FCFS(C) achieves within 1% of the average execution time as compared to FR-FCFS(O). We also evaluated the execution time of Row Fetch, which is a variation of the ReTagger architecture. Similar to ReTag, Row Fetch monitors the scheduling queue to detect a no outstanding signal. However, it transfers the entire row from the HBM layers to the cache controller rather than tag bits only; therefore, it imposes a significant bandwidth overhead to the HBM interface. The motivation for modeling Row Fetch is to assess the performance potentials of accelerating the "row miss, tag match" read access in Figure 3. We observe that Row Fetch results in significant performance degradation compared to ReTag.

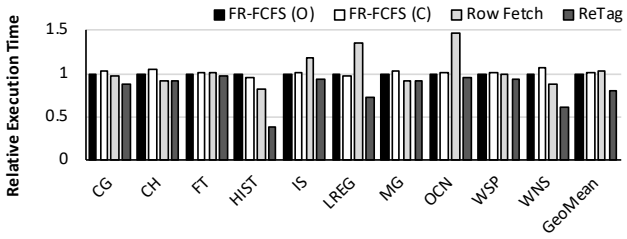

Figure 8: The relative execution times of the evaluated applications using the proposed and baseline cache controllers.

System Energy. The performance improvements gained by ReTagger translates to a reduction in the system energy due to (1) reducing the time and consequently static energy in the processor, HBM cache, and main memory, (2) reducing the total number of refresh operations in all DRAM arrays due to a faster execution, and (3) eliminating unnecessary accesses to HBM for the "row miss, tag mismatch" and "row hit, tag match" writes (Section 3). Figure 9 shows the system energy consumed by the proposed and baseline architectures when executing the evaluated benchmark applications. ReTagger achieve an average of 24% reduction in the system energy, which is the highest saving compared to all of the evaluated DRAM cache interfaces.

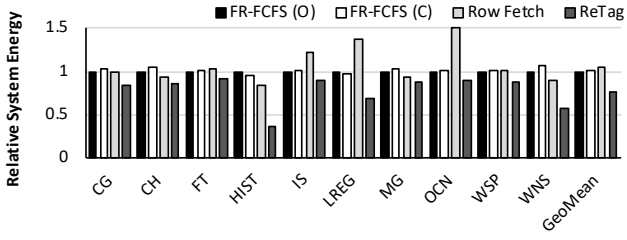

Figure 9: The relative system energy of the evaluated systems using the proposed and baseline cache controllers.

6 CONCLUSION

3D die-stacking has enabled energy-efficient solutions for near data processing by integrating multiple dice of high-density memory layers and processor cores within the same package. This paper presented a novel design approach to HBM cache controllers that relies on monitoring the cache traffic at run time and optimizing the HBM command schedule for better bandwidth and energy-efficiency. We evaluated the proposed mechanism on a set of ten

data-intensive applications that indicate an average performance improvement of 20% over the state-of-the-art DRAM architectures.

ACKNOWLEDGMENTS

The authors would like to thank anonymous reviewers for useful feedback. This work was supported in part by the National Science Foundation (NSF) under Grant CCF-1755874.

REFERENCES

[1] "The exponential growth of data." https://insidebigdata.com/2017/02/16/the-exponential-growth-of-data/.
[2] J. Ousterhout, C. Kozyrakis, D. Mazières, A. Narayanan, D. Ongaro, M. Rosenblum, S. Rumble, and R. Stutsman, "Ramcloud: Scalable high-performance storage entirely in dram," 2009.
[3] ITRS, *International Technology Roadmap for Semiconductors: 2013 Edition.* http://www.itrs.net/Links/2013ITRS/Home2013.htm.
[4] "The top ten exascale research challenges," *Report of the Advanced Scientific Computing Advisory Committee Subcommittee*, 2014.
[5] J. Bolaria, "Micron reinvents dram memory: New architecture and packaging density and peformance," *Microprocessor Report*, pp. 1–6, 2011.
[6] H. M. C. Consortium, "About hybrid memory cube." http://www.hybridmemorycube.org/technology.html.
[7] M. P. (Intel), *An Intro to MCDRAM (High Bandwidth Memory) on Knights Landing.* Intel, January 2016. https://software.intel.com/en-us/blogs/2016/01/20/an-intro-to-mcdram-high-bandwidth-memory-on-knights-landing.
[8] J. Ahn, S. Yoo, O. Mutlu, and K. Choi, "Pim-enabled instructions: A low-overhead, locality-aware processing-in-memory architecture," in *Computer Architecture (ISCA), 2015 ACM/IEEE 42nd Annual International Symposium on*, pp. 336–348, IEEE, 2015.
[9] D. Kim, J. Kung, S. Chai, S. Yalamanchili, and S. Mukhopadhyay, "Neurocube: A programmable digital neuromorphic architecture with high-density 3d memory," in *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*, pp. 380–392, IEEE, 2016.
[10] K. Hsieh, E. Ebrahimi, G. Kim, N. Chatterjee, M. O'Connor, N. Vijaykumar, O. Mutlu, and S. W. Keckler, "Transparent offloading and mapping (tom): Enabling programmer-transparent near-data processing in gpu systems," in *Proceedings of the 43rd International Symposium on Computer Architecture*, pp. 204–216, IEEE Press, 2016.
[11] S. Khan, C. Wilkerson, Z. Wang, A. R. Alameldeen, D. Lee, and O. Mutlu, "Detecting and mitigating data-dependent dram failures by exploiting current memory content," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 27–40, ACM, 2017.
[12] C. Chou, A. Jaleel, and M. K. Qureshi, "Bear: Techniques for mitigating bandwidth bloat in gigascale dram caches," in *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, pp. 198–210, June 2015.
[13] V. Young, P. J. Nair, and M. K. Qureshi, "Dice: Compressing dram caches for bandwidth and capacity," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ISCA '17, (New York, NY, USA), pp. 627–638, ACM, 2017.
[14] "Silicon interposer design: Architecture through implementation."
[15] "Wide i/o 2 (wideio2)." http://www.jedec.org/standards-documents/results/jesd229-2.
[16] S. Rixner *et al.*, "Memory access scheduling," in *Proceedings of the 27th annual international symposium on Computer architecture*, May 2000.
[17] I. Hur and C. Lin, "A comprehensive approach to dram power management.," in *HPCA'08*, pp. 305–316, 2008.
[18] O. Mutlu and T. Moscibroda, "Parallelism-aware batch scheduling: Enhancing both performance and fairness of shared dram systems," in *in Proceedings of the 35th Annual International Symposium on Com- puter Architecture*, pp. 32–41, ACM Press, 2008.
[19] Y. Kim, D. Han, O. Mutlu, and M. Harchol-Balter, "Atlas: A scalable and high-performance scheduling algorithm for multiple memory controllers," in *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, pp. 1 –12, jan. 2010.
[20] E. Ebrahimi, R. Miftakhutdinov, C. Fallin, C. J. Lee, J. A. Joao, O. Mutlu, and Y. N. Patt, "Parallel application memory scheduling," in *The 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-44)*, December 2011.
[21] Y. Kim, M. Papamichael, O. Mutlu, and M. Harchol-Balter, "Thread cluster memory scheduling: Exploiting differences in memory access behavior," in *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '43, (Washington, DC, USA), pp. 65–76, IEEE Computer Society, 2010.
[22] B. Diniz, D. O. G. Neto, W. M. Jr., and R. Bianchini, "Limiting the power consumption of main memory," in *ISCA*, pp. 290–301, 2007.
[23] H. Zheng, J. Lin, Z. Zhang, E. Gorbatov, H. David, and Z. Zhu, "Mini-rank: Adaptive dram architecture for improving memory power efficiency," in *Microarchitecture, 2008. MICRO-41. 2008 41st IEEE/ACM International Symposium on*, pp. 210 –221, nov. 2008.
[24] C. Isen and L. John, "Eskimo - energy savings using semantic knowledge of inconsequential memory occupancy for dram subsystem," in *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, pp. 337 –346, dec. 2009.
[25] K. Sudan, N. Chatterjee, D. Nellans, M. Awasthi, R. Balasubramonian, and A. Davis, "Micro-pages: increasing dram efficiency with locality-aware data placement.," in *ASPLOS'10*, pp. 219–230, 2010.
[26] M. N. Bojnordi *et al.*, "Pardis: A programmable memory controller for the ddrx interfacing standards," in *ISCA*, 2012.
[27] J. Stuecheli, D. Kaseridis, H. C. Hunter, and L. K. John, "Elastic refresh: Techniques to mitigate refresh penalties in high density memory," in *MICRO*, pp. 375–384, 2010.
[28] S. Liu, K. Pattabiraman, T. Moscibroda, and B. G. Zorn, "Flikker: saving dram refresh-power through critical data partitioning.," in *ASPLOS* (R. Gupta and T. C. Mowry, eds.), pp. 213–224, ACM, 2011.
[29] M. K. Qureshi and G. H. Loh, "Fundamental latency trade-off in architecting dram caches: Outperforming impractical sram-tags with a simple and practical design," in *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 235–246, IEEE Computer Society, 2012.
[30] G. H. Loh and M. D. Hill, "Efficiently enabling conventional block sizes for very large die-stacked dram caches," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 454–464, ACM, 2011.
[31] C. Chou, A. Jaleel, and M. K. Qureshi, "Bear: techniques for mitigating bandwidth bloat in gigascale dram caches," in *Computer Architecture (ISCA), 2015 ACM/IEEE 42nd Annual International Symposium on*, pp. 198–210, IEEE, 2015.
[32] P. Behnam, A. P. Chowdhury, and M. N. Bojnordi, "R-cache: A highly set-associative in-package cache using memristive arrays," in *2018 IEEE 36th International Conference on Computer Design (ICCD)*, pp. 423–430, IEEE, 2018.
[33] D. Jevdjic, G. H. Loh, C. Kaynak, and B. Falsafi, "Unison cache: A scalable and effective die-stacked dram cache," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 25–37, IEEE Computer Society, 2014.
[34] Y. Lee, J. Kim, H. Jang, H. Yang, J. Kim, J. Jeong, and J. W. Lee, "A fully associative, tagless dram cache," in *ACM SIGARCH Computer Architecture News*, vol. 43, pp. 211–222, ACM, 2015.
[35] H. Jang, Y. Lee, J. Kim, Y. Kim, J. Kim, J. Jeong, and J. W. Lee, "Efficient footprint caching for tagless dram caches," in *High Performance Computer Architecture (HPCA), 2016 IEEE International Symposium on*, pp. 237–248, IEEE, 2016.
[36] X. Yu, C. J. Hughes, N. Satish, O. Mutlu, and S. Devadas, "Banshee: Bandwidth-efficient dram caching via software/hardware cooperation," *arXiv preprint arXiv:1704.02677*, 2017.
[37] V. Young, C. Chou, A. Jaleel, and M. Qureshi, "Accord: Enabling associativity for gigascale dram caches by coordinating way-install and way-prediction," in *ISCA*, IEEE, 2018.
[38] "Free PDK 45nm open-access based PDK for the 45nm technology node." http://www.eda.ncsu.edu/wiki/FreePDK.
[39] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *International Symposium on Computer Architecture*, 2009.
[40] N. P. Jouppi *et al.*, "Cacti-io: Cacti with off-chip power-area-timing models," *TVLSI*, 2015.
[41] "Micron DDR4 power calculator." DDR4SDRAMSystem-PowerCalculator(xlsm) -Micron.
[42] "Micron LPDDR3 power calculator." http://www.micron.com/.
[43] K. Chandrasekar *et al.*, "Drampower: Open-source dram power & energy estimation tool," *URL: http://www. drampower. info*, 2012.
[44] M. O'Connor, N. Chatterjee, D. Lee, J. Wilson, A. Agrawal, S. W. Keckler, and W. J. Dally, "Fine-grained dram: energy-efficient dram for extreme bandwidth systems," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 41–54, ACM, 2017.
[45] E. K. Ardestani and J. Renau, "ESESC: A Fast Multicore Simulator Using Time-Based Sampling," in *International Symposium on High Performance Computer Architecture*, HPCA'19, 2013.
[46] R. M. Yoo *et al.*, "Phoenix rebirth: Scalable mapreduce on a large-scale shared-memory system," in *IISWC*, 2009.
[47] D. H. Bailey *et al.*, "The nas parallel benchmarks," in *HPCA*, 1991.
[48] S. C. Woo *et al.*, "The splash-2 programs: Characterization and methodological considerations," in *ISCA, IEEE*, 1995.