History-Assisted Adaptive-Granularity Caches (HAAG$) for High Performance 3D DRAM Architectures

Ke Chen[†] Sheng Li[‡] Jung Ho Ahn[§] Naveen Muralimanohar[¶] Jishen Zhao[#] Cong Xu[♭]
Seongil O[§] Yuan Xie[♮] Jay B. Brockman[⊤] Norman P. Jouppi[⊥]

[†]Oracle Corporation[*] [‡]Intel Labs[*] [§]Seoul National University
[#]University of California at Santa Cruz[*] [¶]Hewlett-Packard Labs [♭]Pennsylvania State University
[♮]University of California at Santa Barbara [⊤]University of Notre Dame [⊥]Google[*]

[†]ke.c.chen@oracle.com [‡]sheng.r.li@intel.com [§]{gajh, swdfish}@snu.ac.kr
[¶]naveen.muralimanohar@hp.com [#]jishen.zhao@ucsc.edu [♭]czx102@psu.edu
[♮]yuanxie@ece.ucsb.edu [⊤]jbb@nd.edu [⊥]jouppi@acm.org

ABSTRACT

3D-stacked DRAM has the potential to provide high performance and large capacity memory for future high performance computing systems and datacenters, and the integration of a dedicated logic die opens up opportunities for architectural enhancements such as DRAM row-buffer caches. However, high performance and cost-effective row-buffer cache designs remain challenging for 3D memory systems. In this paper, we propose History-Assisted Adaptive-Granularity Cache (HAAG$) that employs an adaptive caching scheme to support full associativity at various granularities, and an intelligent history-assisted predictor to support a large number of banks in 3D memory systems. By increasing the row-buffer cache hit rate and avoiding unnecessary data caching, HAAG$ significantly reduces memory access latency and dynamic power. Our design works particularly well for manycore CPUs running (irregular) memory intensive applications where memory locality is hard to exploit. Evaluation results show that with memory-intensive CPU workloads, HAAG$ can outperform the state-of-the-art row buffer cache by 33.5%.

Categories and Subject Descriptors

B.3.1 **[Memory Structures]**: Semiconductor Memories

General Terms

Design; Performance; Experimentation

Keywords

3D DRAM; HAAG$; row buffer cache; adaptive granularity

∗ This work was done while Ke Chen was at the University of Notre Dame, and while Norman P. Jouppi, Sheng Li, and Jishen Zhao were with Hewlett-Packard Labs.

ICS'15, June 8–11, 2015, Newport Beach, CA, USA.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-3559-1/15/06 ...$15.00.
DOI: http://dx.doi.org/10.1145/2751205.2751227 .

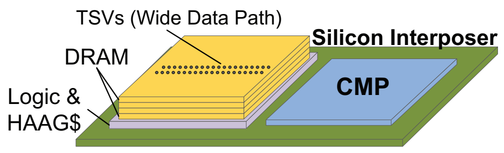


Figure 1: The target system design for integrating the HAAG$: a hybrid 3D DRAM stack and a chip multiprocessor are co-located on a silicon interposer.

1. INTRODUCTION

Future high performance computing demands high performance and large capacity memory subsystems with low cost. Existing mainstream DDR3/4 DRAM systems, however, are neither efficient nor capable of meeting these demands because of limited pin count per socket [31] and fan-out restrictions in high-speed signaling [30, 33]. Emerging 3D stacked memory [14, 17] is a promising solution to provide high bandwidth (with high bank concurrency), large memory capacity (with multiple stacked DRAM die in a package), and cost efficiency (by sharing common peripheral circuitry over the stack). Moreover, the hybrid integration of logic die into DRAM stack [29] (as shown in Figure 1) also opens up opportunities for architecture enhancements in memory.

In conventional DRAM, row buffers are employed to keep a data row open (active) so as to avoid long latency when accessing the large DRAM cell array repetitively for contiguous memory references (i.e., with adjacent addresses). With a desire to maintain multiple rows open even when there are row-buffer conflicts, a row-buffer cache (i.e., cache of row buffers) [9, 15, 19, 37] has been proposed to further improve performance. By caching memory data with high access locality on the memory side, a DRAM row-buffer cache can help lower the average memory reference latency, and reduce the number of power-hungry row-buffer activations and precharges in DRAM. Despite these potential benefits, previous row-buffer cache designs suffer from implementation challenges when being fabricated on the same DRAM die, especially due to the cost constraints such as limited metal layers, slow DRAM process, and tight area budget.

Emerging hybrid memory stacks with both memory layers and a logic layer (Figure 1) are ideally suited for DRAM row-buffer caches. First, a row-buffer cache can now be fabricated on the separate logic layer with a logic process that can be optimized for high performance and power efficiency. Second, a vertical dimensional interconnect – through silicon vias (TSVs) – is introduced to shorten the data transfer latency between the row buffers in stacked DRAM and the row-buffer cache on the logic die.

However, designing efficient row-buffer caches for 3D memory systems remains challenging, with one reason being the trade-off between cost and efficiency (performance and energy). In particular, although ideally TSVs can have a high density and bandwidth with their small size, the data bus width between the stacked die is still constrained. This is because increasing the data bus width incurs large routing logic area overhead and therefore increases the cost significantly. Figure 2 illustrates the cost increase versus data bus width (see Section 4 for our detailed methodology). As seen in the figure, the area overhead can be more than 20% higher for memory data buses that are 512b/bank or wider, which is prohibitively high for extremely cost-sensitive memory products.

As a result, existing 3D DRAM products such as Micron's Hybrid Memory Cubes (HMC) [29] (highlighted in Figure 2) and Samsung's 3D DRAM memory [14] have adopted a conservative internal data bus width, which is far smaller than the size of an entire DRAM row. The resulting limited internal memory data paths inside the 3D memory stack have made it impractical to transfer the entire DRAM row every time (e.g., 256 cycles with a 64-bit data bus for 2KB DRAM row). As also pointed out in [23], this mismatch between internal data bus width and DRAM row size leads to a limited performance benefit for a conventional row-buffer cache because of contention on the internal narrow data bus and timing constraints imposed by modern DRAM technologies, even for 3D memory systems.

The recent "register-file approach" row-buffer cache (FM-RB\$) proposed by Loh [23] tries to address this challenge and achieves better performance on 3D memory systems with constrained internal/TSV channel width by selectively caching DRAM rows. However, it still caches the entire row every time regardless of the program locality and the limited TSV channel width, resulting in unnecessary performance and energy losses. Also, the FM-RB\$ design uses a pending memory request queue in the memory controller as decisive information for caching. Given the large number of banks (e.g., 128 [29]) in modern high-performance 3D DRAMs, the narrow window of a per-bank memory controller request queue can suffer from limited visibility to memory access locality and leads to sub-optimal caching decisions. Because of these limitations, the demonstrated gain in performance and energy-efficiency of FM-RB\$ design was mainly from GPU workloads with very regular memory access patterns and high locality, whereas supporting a wider range of applications remains an opportunity for new innovations.

In order to tackle these challenges and to fully unleash the performance and energy efficiency potential of 3D DRAM systems, especially in manycore CPU systems running memory intensive applications where memory-side access locality is hard to exploit [2], we propose a History-Assisted Adaptive-Granularity Cache (HAAG\$). HAAG\$ is the first attempt to design a DRAM row-buffer cache with adaptive cache line granularity and an intelligent data mapping scheme. The goal is to efficiently utilize the narrow TSV channel of a 3D DRAM memory system, and maximize performance and energy efficiency for data-intensive applications even with irregular memory access patterns (as usually found in multithreaded applications). The innovative features of our HAAG\$ include:

- An intelligent predictor that decides: 1) whether to cache the corresponding DRAM row for an outstanding request, and 2) if so, how much of the DRAM row should be cached. The predictor not only increases the row-buffer hit rate with more accurate prediction, but its queue size is also scalable to keep up with the increasing number of banks in 3D stacked memory;

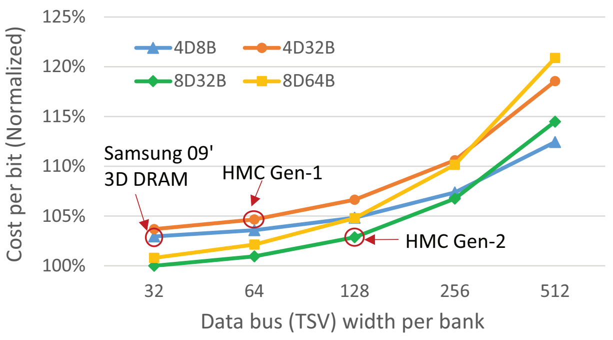


Figure 2: Cost for 3D DRAM stack configurations with different die counts in a stack ('D') , bank counts in a die ('B'), and memory bus width (or equally the TSV channel width). Several typical 3D DRAM configurations such as Samsung's 3D DRAM memory [14] and Micron's HMCs (both Gen1 and Gen2) [29] are highlighted.

- An adaptive-granularity cache that dynamically adjusts the amount of data to cache for different DRAM accesses. One novelty is a power efficient content addressable memory (CAM) design with an improved tag array that enables mapping to cache lines with variable granularities. It improves performance and power by avoiding transferring unnecessary data while increasing the data storage efficiency.

Figure 1 also shows the target system design for a 3D DRAM stack equipped with a HAAG\$ which serves a manycore processor. It co-locates the stack on the same silicon interposer as the chip multiprocessor (CMP), similar to either the near-memory HMC design [29] or the JEDEC High Bandwidth Memory (HBM) standard [12][1] (with a logic layer below the HBM stack). In other words, a HAAG\$ design can be flexibly incorporated into DRAM stack architecture and is compatible with multiple interfaces. Our system-level evaluation results show that our HAAG\$ design outperforms the state-of-the-art by 33.5% for memory-intensive CPU applications.

2. MOTIVATION AND RELATED WORK

This section first reviews the related work on 3D memory architecture and motivates designing sophisticated DRAM row-buffer caches. Then we present 3D-stacked DRAM structures where the design and optimization of a DRAM row-buffer cache is tightly coupled with the design of the 3D DRAM.

2.1 Row-Buffer Cache in 3D Memory

While there have been recent studies on 3D memory that focus on stacking memory die directly on a processor die to enable fast access to DRAM [6, 13, 16, 22, 24, 34], there is a relative absence of research on more conventional 3D memory systems. The first commercial products are 3D stacked DRAM [14, 17] (from Samsung) and hybrid memory cubes [1, 29] (from Micron). Although these products have shown superior performance and power efficiency compared to conventional DRAMs, further gains are possible only by fully exploring 3D memory's potential advantages at the architecture level.

The row-buffer cache is a promising architectural enhancement to leverage the integrated logic die of hybrid memory stacks. Although row-buffer caches and cached DRAM [9, 15, 37] have been

[1]HBM is an emerging DRAM interface standard for 3D-stacked DRAM defined by the JEDEC organization, which uses a wide-interface architecture to achieve high-speed, low-power operation.

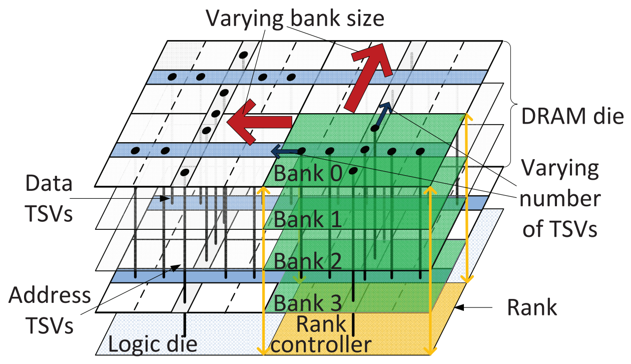


Figure 3: Illustration of hybrid DRAM stack internal structure, including rank, bank, and layout of peripheral circuitry and TSVs.

proposed to overcome the problem of row-buffer conflicts in memory, the cost of modifying the existing bank structure to support additional buffers has kept memory manufacturers from embracing this technique. With the added logic die of a hybrid memory stack and the silicon area it provides, it is now feasible to support such extensions without significantly increasing the overall memory cost.

However, a basic row-buffer cache that caches DRAM data on every memory access only results in marginal performance gain [23] because the row-buffer cache is a limited resource with high competition. In addition, the problem with deploying row-buffer caches in stacked memory is that data from DRAM cells has to be sent through cost-constrained and thus relatively narrow TSV channels (e.g., 64-bit wide [29]), which is detrimental to performance and power as every activation now requires multiple cycles to send the entire page to the row-buffer cache (e.g., 256 cycles for 2KB DRAM row).

FM-RB$ proposed by Loh [23] attempts to address the high overhead in a basic row-buffer cache design from caching every activated page. It looks ahead into the pending memory request queue in a memory controller to decide whether to allocate an entry in the row-buffer cache or to access the data directly from a bank's row buffer. However, it still caches the entire DRAM row despite the narrow TSV data bus available and the limited memory access locality over an entire DRAM page. This wastes time, power, and storage by loading and evicting unused data on the data bus, meanwhile limiting the performance and energy benefits for applications with different memory access granularity requirements. As a result, although FM-RB$ demonstrated significant performance gain on GPU workloads (with very high locality), it showed limited performance gain for CPU applications because they have low locality which requires extra efforts to leverage. Our HAAG$ is designed to solve this challenge for CPU workloads.

Our proposed HAAG$ for 3D DRAM is better in several ways than the state-of-the-art FM-RB$ design. First, the cache line granularity can vary dynamically to adapt to applications' requirements at runtime, which is especially beneficial for irregular access patterns. Second, while FM-RB$ only considers the pending memory requests to identify targets for row-buffer caches, the proposed HAAG$ deploys a decoupled predictor queue that also holds the addresses of previously served memory requests. This decoupled predictor queue not only increases the prediction accuracy with extended information of the historical memory access pattern, but is also scalable to meet prediction window size requirements at low hardware cost for 3D memory with a large number of banks.

2.2 Hybrid Memory Stack Structure

Stacking a base logic die under 3D DRAM opens up the opportunity to make the DRAM row-buffer cache achieve high performance and cost efficiency, and tightly couples the configuration of the DRAM cache design with the 3D DRAM architecture. Thus it is important to understand the 3D DRAM structure and its implications on row-buffer cache designs.

Figure 3 shows the internal structure of a typical hybrid DRAM stack that will be used in the study of HAAG$. DRAM die are stacked on top of each other, while a set of vertically overlapped banks in the stack can be grouped together and function as a rank. In a typical memory setup, a rank consists of a set of DRAM banks that share the same memory bus (in this case TSVs) to the memory controller. While each bank within a rank can operate concurrently, at any given time only one bank can communicate to the memory controller through the shared bus.

For stacked DRAM, the bank internal organization is similar to that of single-layer DRAM because of the TSV pitch mismatch with on-chip wires and a desire to avoid modifications to the basic building blocks. A bank can consist of one or multiple DRAM cell arrays. As sharing reduces the footprint of peripheral circuits of 3D DRAM, a larger number of smaller banks can be employed than conventional planar DRAM, resulting in better concurrency, reduced bank access latency, and more effective use of TSVs as internal buses. A similar design can be found in the Micron HMC [29] and Microbank [33].

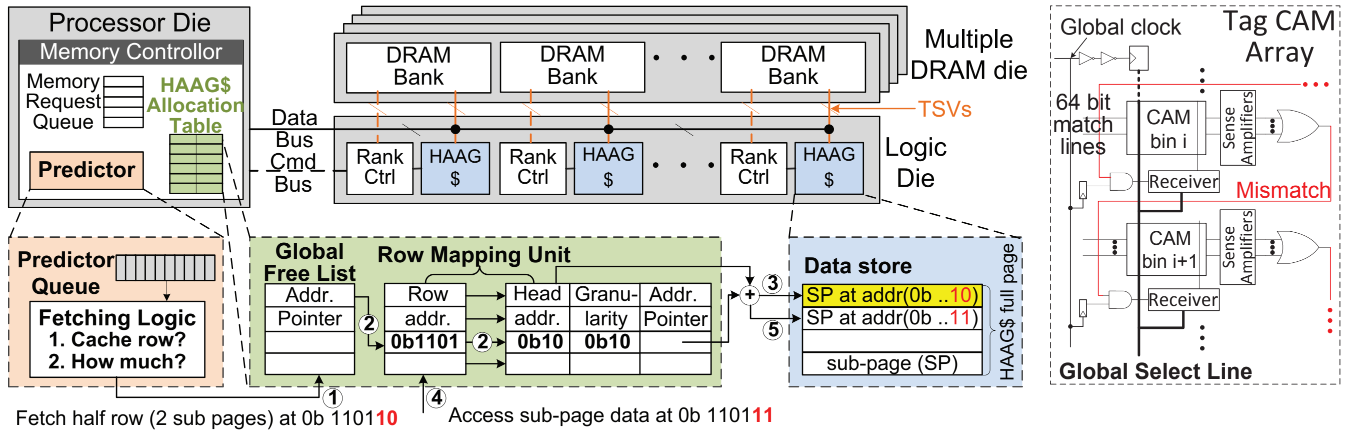
TSVs are laid out on the edge of DRAM arrays of a bank, and used as the inter-die data/address/command bus to connect the circuit blocks on the base logic die to the DRAM banks above. As mentioned in Section 1, the TSV data bus width is constrained by the bank internal datapath, and further limits the internal data transfer bandwidth. This raises the question of how to best leverage the internal data bandwidth in a DRAM stack, and is a key aspect of HAAG$.

The dedicated logic die holds common peripheral circuit blocks that can be shared among DRAM die to reduce cost and energy. This includes rank control logic, input buffers, delay-locked loop (DLL), and clock circuitry. The existence of various circuit blocks limits the remaining real estate on the logic die, which should be carefully exploited for the integration of HAAG$.

3. HISTORY-ASSISTED ADAPTIVE-GRANULARITY CACHE (HAAG$)

Figure 4a shows an overview of the proposed HAAG$ architecture, where the key components are a scalable predictor with historical memory address information to assist making caching decisions and an adaptive-granularity row-buffer cache with elastic mapping. The predictor is used to decide whether to and how much to cache the corresponding DRAM row for an outstanding memory request. It is placed on the processor side to make fast cache/not-cache decisions by directly accessing existing entries in pending memory request queue and hit/miss information of an allocation table (detailed below) inside the memory controller. The predictor also has access to historical memory request addresses stored in predictor queue (detailed below) to obtain more comprehensive information on memory access locality to make more accurate caching decisions for each DRAM bank.

Meanwhile, the adaptive-granularity row-buffer cache consists of a data store and an allocation table that keeps the meta information (e.g., granularity and head address) of the rows in the data store. While both the allocation table and data store logically belong to the same row-buffer cache, they are physically separated



(a) Overview of the History-Assisted Adaptive-Granularity Cache (HAAG$) architecture and operation flow. (b) CAM in row mapping unit.

Figure 4: Illustration of (a) the History-Assisted Adaptive-Granularity Cache (HAAG$) architecture and an example flow of partial row caching and a subsequent hit, and (b) a specially designed CAM design for the tag portion in the row mapping unit. The HAAG$ is composed of a predictor and an allocation table in the memory controller as well as a data store on the logic die.

to maximize performance benefits and achieve easy implementation with existing memory interfaces and protocols. The allocation table (i.e., tag array) of the HAAG$ is placed close to the processor-side memory controller to avoid long round-trip latency for probing the cache status if it were implemented on the memory-side logic die. The data store of a HAAG$ is located on the logic die below the DRAM stack to directly receive/send data from/to the DRAM row buffers through TSVs, and the allocation table is placed in the processor side as part of the memory controller to help make fast caching decisions and detecting hits and misses.

The memory controller consults the allocation table and the predictor before scheduling a memory request, and then issues appropriate command sequences based on the results of hit or miss on the HAAG$, and the HAAG$ caching or not caching decision for the pending memory request. This design is easy to implement as it is compatible with the design methodology of current memory interfaces (e.g., DDR3/4) where the memory controller is the master device and maintains deterministic timing of memory commands and data replies.[2] Accordingly, the DRAM and the HAAG$ share common command/address/data buses. Note that HAAG$ is a lower level memory hierarchy than the memory controller. Thus it is a serialization point and does not need to maintain coherency. We will describe the implementation details of these hardware components, and demonstrate their function and usage through an example operation flow in the following sub-sections.

3.1 History-Assisted Scalable Predictor

The dedicated predictor shown in Figure 4a consists of two parts: a **Decoupled Predictor Queue** and **Fetching Logic**.

The Decoupled Predictor Queue: The prediction of whether to cache data to the row-buffer cache as well as what part of the row to cache is based on the memory locality. The pending memory request queue inside a memory controller could be used for prediction, and for making decisions to allocate an entry into the row-buffer cache when there are future references for that particular row. However, the memory request queue in the memory controller is a very narrow window (e.g., 32 entries), and provides very limited visibility to memory locality. While this approach works well for memories with a limited number of banks per channel (16 as assumed in previous work [23]), it is not scalable to satisfy the demands of up to hundreds of banks per channel in near-future 3D DRAM designs [29], which are employed to mitigate the notorious row-buffer conflict problems [2] and to increase parallelism. First, a large bank count leads to high bank-level parallelism, and therefore reduces the number of queued-up pending memory requests (i.e., prediction window size) per bank. As a result, there is insufficient access pattern information in the pending memory request queue, significantly lowering the prediction accuracy of the row-buffer cache data caching. Second, increasing the size of the pending memory request queue is very costly [4], because it holds all memory access information including addresses, data, and control in a complex and power-hungry content addressable memory (CAM) structure.

The Decoupled Predictor Queue is proposed to meet scalability demands, and to solve the problems of limited memory access pattern availability and large overhead of a long pending memory request queue. The Decoupled Predictor is a separate hardware queue in the memory controller that stores the addresses of both the pending memory requests *and* previously served requests. The queue size is more scalable than in previous approaches because there can be as many served memory request entries in the queue as the system demands. Further, the access pattern information is more comprehensive, not only because of a larger queue size, but also because of a combination of *future* and *historical* memory accesses in the predictor queue. Finally, the hardware overhead is much less than the pending memory request queue with the same number of entries in that it only needs to store addresses in the queue. In summary, the Decoupled Predictor Queue is much more capable to meet the demands on increasing predictor queue size for a large number of banks in 3D memory. The flexible addition of historical memory requests provides a much wider information window and thus more visibility into memory access locality than only the MC pending request queue with limited entries, and there-

[2]For systems with decoupled/disintegrated memory controllers [8] where the processor-side memory controller only packetizes memory requests and off-loads the protocol implementation (e.g., DDRx, LPDDRx) to memory-side co-controllers, both the allocation table and data store of the HAAG$ can be located at the memory side co-controller.

fore significantly increases the caching prediction accuracy as we will show in Section 5.

The Fetching Logic is implemented to make selective caching decisions with adaptive granularities based on the information stored in the predictor queue. Specifically, the fetching logic makes two decisions for each outstanding memory request based on the predictor queue: 1) whether to cache data from the same row as the outstanding request, and 2) if so, how much data on adjacent addresses needs to be cached.

The first decision, *whether* to cache a DRAM row to the HAAG$ on the logic die is made by techniques similar to [23] "Load with Modification"[3], "Few Uses"[4], and "Bank-aware Deallocation"[5]. If the fetching logic decides to fetch a DRAM row and cache it in the HAAG$, then in addition it makes a second decision as *how much* data (or what fraction) of the DRAM row should be cached. This step is unique to HAAG$ and essential for improving performance and energy efficiency in the context of limited TSV bus width in practical 3D architectures. This second decision is made by comparing all the memory requests in the predictor queue with the target row address of the outstanding request. Assuming that the size of a CPU last-level cache line is 64 bytes, the fraction of the DRAM row (i.e., how many 64 byte data blocks) to be fetched and cached is determined by the longest distance between requested cache lines (including both pending and past requests) in the same row.

Since 64-byte lines are cached in the processor LLC anyway and 128-byte data can also be cached in the processor by next line prefetching, caching data blocks smaller than 256 bytes in the HAAG$ can hardly provide a significant performance gain. Thus, the predictor chooses to cache data with adaptive granularity from a *sub-page* of size 256B to a *full page* with the size of an entire DRAM row, in increments of a sub-page. We use a simple pop count circuit to tally the number of row address matches among the requests in the predictor queue and pending request queue, and all the sub-pages with the same row address as the outstanding memory request will be cached. Here the final decision made by the fetching logic is sent via the command bus to the rank controllers in the 3D DRAM stack as shown in Figure 4a. Since the internal CPU clock is faster than an external memory transaction, this approach can be easily implemented with minimal overhead. For example, if a DRAM row has 16 CPU LLC lines and is partitioned into four sub-pages (each has four CPU cache lines), and the 1st, 4th, and 9th CPU cache line in the row are predicted as "useful", then the corresponding 1st, 2nd, and 3rd sub-pages will be fetched.

3.2 Adaptive-Granularity Row Buffer Cache

Since multithreaded and multiprogramming applications running on multi/many-core systems do not demonstrate distinct memory access granularity preferences [21, 36], a fixed row-buffer cache line size can only improve performance on a few applications while degrading performance for others. Since different cache line sizes benefit different benchmarks, only a row buffer cache with adaptive granularity can improve performance for all applications. Prior work [11] also sought to dynamically optimize cache line sizes by exploiting the memory reference locality; but the target was on-chip caches with embedded DRAM, and the scheme for determining variable sizes of cache lines is different.

Being a small and scarce resource while supporting different access granularities from a sub-page to a full page, HAAG$ gives rise to three special design requirements. First, adjacent sub-pages of a DRAM row need to be placed adjacently in the HAAG$ to enable burst block transfers between the HAAG$ and the memory stack. Otherwise, evicting a full page or several sub-pages from the HAAG$ will result in multiple row-buffer activations and precharges that are slow and power-hungry. Second, partial rows need to be placed as close as possible in the HAAG$ to eliminate fragmentation. Finally, the HAAG$ should ideally be fully associative to avoid expensive conflicts that involve lengthy and power-hungry DRAM row activations/precharges. It is challenging to design such a fully-associative adaptive granularity cache at low cost.

Our HAAG$ design is able to meet these requirements and support full associativity at various granularities at a low cost with low fragmentation. Taking a 128KB-sized HAAG$ as an example, its associativity is 512 with access granularity of a 256B sub-page, and 64 with a 2KB full page. Figure 4a illustrates the adaptive-granularity fully associative HAAG$ architecture, which consists of two main parts: the *HAAG$ allocation table* and the *data store*. To reduce the hardware cost from supporting the highest adaptive associativity, we design a special content addressable memory (CAM) as shown in Figure 4b (detailed later in this section).

HAAG$ Allocation Table is essentially a specially designed tag array of a common cache, where the HAAG$ allocation table and data store are physically separated and located on the processor-side memory controller and the memory stack respectively. On one side, the HAAG$ allocation table is implemented in the processor-side memory controller to better track which (partial) rows are currently stored in the HAAG$, and therefore accelerate HAAG$ hit and miss detection. Also, the allocation table helps the memory controller with command scheduling. On the other side, the data store of HAAG$ is placed on the logic layer and close to the 3D-stacked DRAM to directly use TSV buses for transferring data between stacked DRAM and logic die. This split cache design is reasonable particularly because DRAM accesses are less latency sensitive than the small caches on processor chips, and the miss rates are typically higher.

The HAAG$ allocation table is the heart of the adaptive-granularity fully associative cache. There are two major components in the HAAG$ allocation table: a *global free list* and a *row mapping unit*:

- *The global free list* holds occupancy information for the cache lines in the data store, and is responsible for mapping rows in the row mapping unit during cache line install or replacement.

- *The row mapping unit* is a self-contained fully associative cache. The tag portion is implemented in a specially designed CAM, and stores the row addresses of memory blocks for lookup. The data portion holds the cache line information including *head address*, *granularity*, and *address pointers* to index the data store, in addition to common cache information such as dirty bit and valid bit (not shown in the figure).

Data Store. The HAAG$ data store is distributed rather than centralized on the logic die to avoid conflicts between DRAM ranks, and to avoid the energy overhead for transferring data over the long inter-rank data lines. As shown in Figure 4a, each memory rank owns a separate data store that is shared by the stacked DRAM banks within the rank. Multiple adjacent sub-pages can be bundled to form a cache line with larger granularity up to a full page. In this

[3] A DRAM row that is loaded into a HAAG$ entry and written to will eventually be written back to the DRAM row buffer.

[4] A DRAM row with few (or no) additional pending requests in the predictor queue will simply bypass the HAAG$ and directly serve the access from the memory controller.

[5] During cache entry replacement, a victim HAAG$ entry that maps to a different bank than the incoming DRAM row is prioritized to avoid conflicts on the data bus.

process, the sub-pages must be aligned at the corresponding address boundary of the access granularity. It is not possible to have multiple sub-pages from the same DRAM row scattered in the data store. Instead they must be stored contiguously to avoid multiple evictions to the same row.

CAM Design in the Row Mapping Unit. To reduce the cost of maintaining the high associativity of HAAG$ with variable granularities (e.g., from whole row to a sub-page), we design a power efficient CAM for the tag portion of the row mapping unit as shown in Figure 4b. The two key design features are hierarchical searching and hardware supported sorting. The combination of these two techniques ensures that the tags of frequently accessed cache lines (even with different granularities) are placed near the top. Therefore, search operations will usually complete much earlier than in a traditional CAM, reducing energy and latency overhead.

Our hierarchical searching technique divides the CAM array into *bins*, where only the mismatch signal of the previous bin will trigger an access (i.e., comparison) in the next bin. The design is based on a key observation that HAAG$ is different from caches or TLBs that require low access latency and a uniform access for all entries. Essentially, only row addresses are stored in the tag CAM, and the offset information is computed after the row address lookup in the tag CAM. Since different applications usually have certain preferences for memory access granularity [21], row address tags of certain sub-pages will have a much higher hit rate than those of others.

For example, if applications prefer accessing the full page of 2KB in a 64-entry HAAG$ whose sub-page size is 256B, all tag comparisons will happen at the corresponding 64 sub-pages (those are the head of a full page) out of the total 512 in the data store. Therefore, if all these entries are placed in the first bin of the tag CAM in the row mapping unit as shown in Figure 4b, the 512-way associative row mapping unit will have comparable power and timing to a 64-entry CAM, while still covering the worst case.

The memory controller will monitor the average access granularity and periodically sort the tag store by moving the entries with preferred access granularity to upper CAM bins. This hardware sorting can ensure the tags of frequent cache lines will be placed near the top in the common case. The hardware-sorted table can be implemented using techniques in prior work [27] with a time complexity of O(n). Furthermore, the sorting of bins in the CAM can be scheduled to be performed during idle time and may be completed before new memory accesses arrive. It can also pause the sorting by giving higher priority to memory accesses.

Replacement Policy. The Least Recently Used (LRU) replacement policy is not optimal for HAAG$ management because it can cause thrashing of HAAG$ entries. Moreover, if the LRU-selected HAAG$ dirty line is mapped to a busy bank, waiting to evict this line will cause significant delay in servicing pending memory requests. Therefore, a cache line is preferred as the eviction victim if there are no future memory requests to it and it is not mapped to a busy bank. If no HAAG$ lines meet the criteria, the replacement policy naturally degrades to LRU.

In order to support adaptive granularity, we apply two extra rules (in priority order) for eviction victim selection: 1) large enough (equal to or larger than the size of the pending request) cache lines are preferred so that a single eviction will make enough room for the new request, 2) non-dirty cache lines are preferred over dirty lines. If neither of these apply, we use the rules in [23], and prefer a cache line as the eviction victim if it has no future memory requests and is not mapped to a busy bank.

Operation Flow of HAAG$. Figure 4a also demonstrates an example flow of operations in HAAG$.

- *Step 1:* When the predictor detects a new (partial) row that needs to be cached, it sends out requests to the HAAG$. In the example we assume that a full page is 2KB with four sub-pages, and the HAAG$ needs to cache a half row (two sub-pages) with the row address as 0b 1101 (0x D) and the sub-page address as 0b 1101 10.
- *Step 2:* The global free list will pick an available cache line at the 512B boundary in the data store (highlighted in Figure 4a) and insert this into the empty entries of the row mapping unit. The row address and the meta information are stored in the tag and data portions of the row mapping unit respectively. The meta information includes the head address of 0b 10 (the offset address in the DRAM row), the granularity of 0b 10 (two sub-pages), and the pointer to index the data store cache line.
- *Step 3:* The memory controller issues both a DRAM access command and a HAAG$ access command to retrieve the data from DRAM and store the half row (two sub-pages) in the data store.
- *Step 4:* With a new memory request accessing data at sub-page 0b 1101 11, the memory controller consults the HAAG$ allocation table.
- *Step 5:* A hit in the row mapping unit indicates that the (partial) row is cached. Since the granularity of the cached row is unknown at this time, the Hamming distance of this sub-page and that of the head address are compared with the granularity to further confirm whether this is a real hit. In the example, it is a real hit, and the starting address of this half row and the Hamming distance are used together to address the data store for data retrieval.

4. METHODOLOGY

To enable a thorough evaluation of HAAG$ in the context of 3D DRAM, we built a comprehensive evaluation infrastructure consisting of two major components: 1) a modeling framework (shown in Figure 5) to study, design, and optimize HAAG$ considering 3D-stacked memory system cost (including silicon area), energy, and latency; and 2) a simulation framework based on an extended McSimA+ simulator [3] combined with McPAT [20] to evaluate the system-level performance and energy efficiency of HAAG$ with standard SPLASH-2 and SPEC CPU2006 benchmarks.

The design and optimization of HAAG$ depends tightly on the architecture of the 3D DRAM. This makes the design of 3D DRAM and row-buffer cache an inseparable endeavor, and a detailed study of the stacked DRAM design space is a prerequisite for architecting HAAG$. Therefore, we first built a framework as shown in Figure 5 to study the cost, performance, and energy consumption over a large space of 3D DRAM designs to achieve a near-optimal 3D DRAM architecture that drives the final design of the HAAG$.

As memory vendors often compete in commodity markets with thin profit margins, cost takes precedence over other metrics. Thus we first built a comprehensive cost model (Figure 5) based on [7], with die area, die yield, packaging cost and other parameters collected from projections for semiconductor chips using the IC Knowledge[6] models [10], and TSV geometry parameters from ITRS [31] and industrial products [14, 17]. Our cost model covers the cost in the entire manufacturing flow attributed to DRAM fabrication, logic process, and packaging, and also accounts for yield losses

[6] A commercial tool widely used for chip cost projections.

SPLASH-2

Application	Dataset	Application	Dataset
Barnes	16K particles	Cholesky	tk17.O
FFT	1,024K points	Radiosity	room
FMM	16K particles	Raytrace	car
LU	512×512 matrix	Volrend	head
Ocean	258×258 grids		
Radix	8M integers		
Water-Sp	4K molecules		

SPEC CPU2006

Set	Applications
CINT	
high	429.mcf, 462.libquantum, 471.omnetpp, 473.astar
med	401.bzip2, 456.hmmer, 464.h264ref, 483.xalancbmk
low	400.perlbench, 403.gcc, 445.gobmk, 458.sjeng
CFP	
high	433.milc, 450.soplex, 459.GemsFDTD, 470.lbm
med	410.bwaves, 434.zeusmp, 437.leslie3d, 481.wrf
low	436.cactusADM, 447.dealII, 454.calculix, 482.sphinx3

Table 1: SPLASH-2 datasets and SPEC CPU2006 application mixes for high, medium, and low memory bandwidth.

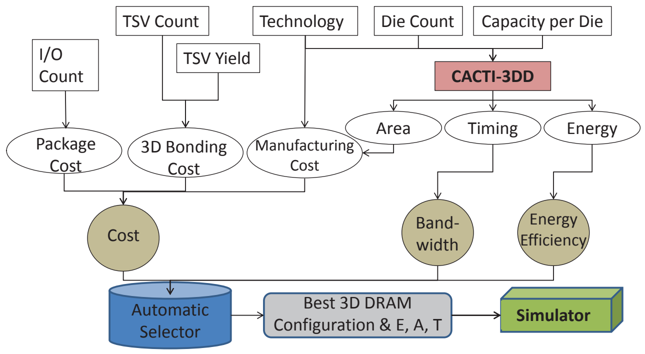


Figure 5: Modeling framework of 3D-stacked DRAM.

during die bonding and TSV fabrication. In particular, chip area and stacking yield are the two major factors that affect DRAM stack cost. We then extended CACTI-3DD [5] to simultaneously model power, area, and timing for 3D-stacked DRAM.

Using this framework (Figure 5) that includes both the extended CACTI-3DD and cost model, we performed a design space exploration for 3D DRAM, where we evaluated the cost, internal data bandwidth, and energy efficiency of the stack. Specifically, the internal data bandwidth is approximately proportional to the bank concurrency and data bus bandwidth, while the energy efficiency is the energy consumption for each memory access divided by the input/output width.

For the system performance study, our simulation infrastructure is based on McSimA+ [3], a Pin [25]-based manycore simulator that models multithreaded in-order and out-of-order cores, caches, directories, on-chip networks, and memory channels in detail. We modified McSimA+ to support HAAG$ with adaptive granularity. We used McPAT [20] to obtain power and energy parameters for our energy efficiency study. We used the SPLASH-2 benchmarks [35] with increased dataset sizes as summarized in Table 1. We also used the SPEC CPU2006 benchmark suite [26] to measure the system performance on consolidated workloads. The benchmark suite consists of both integer (CINT) and floating-point (CFP) benchmarks, all being single-threaded. We picked 12 applications from both CINT and CFP, and made three groups each, with four applications per group, based on their main memory bandwidth demand [2] as shown in Table 1. We found the representative simulation phases of each application and their weights using Simpoint 3.0 [32]. Each hardware thread runs a simulation phase and the number of instances per phase is proportional to its weight. We skipped the initialization phases of each multithreaded workload and simulate two billion instructions unless it finishes earlier.

5. EVALUATION

In this section, we evaluate the performance and energy efficiency improvements of HAAG$ integrated in hybrid 3D DRAM architecture. Figure 6 shows the target system used for evaluating our proposal, which employs a HAAG$ on the logic die of each 3D stack (Figure 6a) and connects the stack to a manycore processor with high-bandwidth interconnect on the same silicon interposer (Figure 6b). The target system closely resembles an HMC near-memory implementation [29] or the JEDEC High Bandwidth Memory (HBM) standard [12] (with an additional logic layer under the HBM stack). A multi-channel memory controller is responsible for communication between the processor and each of the hybrid DRAM stacks.

The manycore processor is organized as multiple clusters of cores interconnected by an on-chip 2D mesh network (Figure 6c). Each cluster has four multithreaded Niagara-like cores [18], and a shared four-bank 16-way L2 cache with a bank size of 256KB (Figure 6d). Each core has four hardware threads and separate 32KB 4-way set associative data and instruction L1 caches. Within a cluster, a crossbar is used for communication between cores and the L2 cache banks. A two-level hierarchical directory-based MESI protocol is used for cache coherency at both L1 and L2 cache levels.

5.1 Architecting the HAAG$ Enabled 3D Memory System

As integrated in the hybrid DRAM stack, the design specifications of a HAAG$, including the capacity, adaptive cache line granularity, and associativity are determined by the organization of the 3D stacked DRAM including the TSV channel width, stack size, and bank count. Also, the 3D DRAM determines a significant portion of the system performance and energy consumption. Thus, before evaluating system-level aspects of the HAAG$, it is necessary to get a reasonable architecture configuration for the 3D-stacked memory and row-buffer cache.

Accordingly, we performed a detailed design space exploration of 3D DRAM architecture and obtained the overall optimal design point accounting for cost, data bandwidth, and energy efficiency. Figure 2 presents a subset of the design exploration results for cost. Specifically, increasing stacked die count is beneficial in terms of area, as the more we stack, the more we will share within a stack. Meanwhile, increasing the memory datapath width causes area overhead and thus adds significantly to the cost. From the overall design space we have the best design point with 8 stacked die, 32 banks per die, and 128 IO per bank, which gives the highest bandwidth (proportional to bank count and TSV data width) and energy efficiency within the acceptable cost overhead toler-

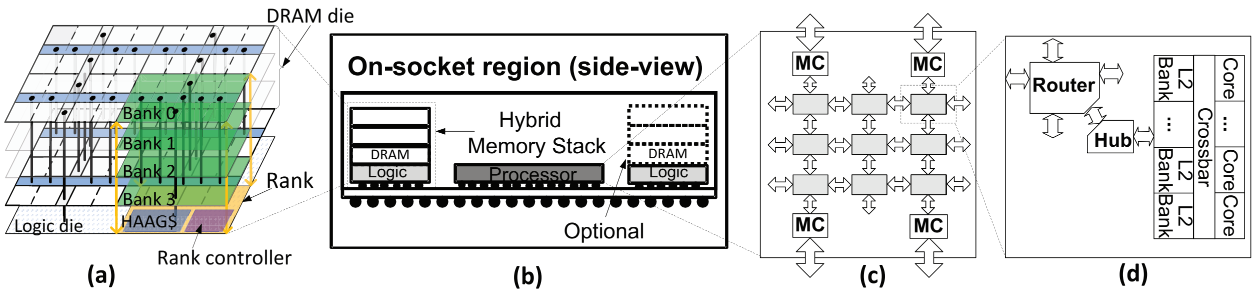


Figure 6: Illustration of our target system architecture in a package: a silicon interposer holding a manycore processor surrounded by hybrid memory stacks with a high-bandwidth interface. (a) A single 3D DRAM stack evaluated in the experiments. (b) Side view of the package. (c) The 2D mesh architecture of the manycore processor with the memory controllers (MC) at the edge. (d) Magnified core architecture.

3D DRAM	Num bank per die 32	IO width per bank 128	Page size 2KB/rank	Read latency 15 ns	Read energy 2.4 nJ	Internal BW 280 GB/s
HAAG$	Capacity per rank 64KB	Associativity* 32 to 256	IO width 512	Read latency† 0.79 ns	Read energy† 0.08 nJ	Access granularity 2KB to 256B
Manycore processor	Thread count 256	Core count 64	Clock 3GHz	L1 32KB 4 Way	L2 256KB/core 16-way	Thermal design power 107.5W

Table 2: Parameters of the manycore processor, 3D-stacked DRAM, and HAAG$ at 22 nm technology. †Worst case numbers for transferring each 64 byte data block to the processor from the HAAG$ including tag lookup and data retrieval (off-chip I/O is not included here). *Depending on access granularity.

ance range (3%). It is worth noting that the design configuration is similar to the second-generation HMC [29].

Table 2 lists the detailed system parameters and modeling results of both the 3D DRAM stack and the HAAG$ at a 22nm technology node. Specifically, the 3D DRAM has a page size of 2KB/rank, and an internal data transfer bandwidth of 280 GB/s. Also, TSVs have a latency of 0.7ns and energy per bit of 0.16 pJ based on CACTI-3DD. The HAAG$ timing and energy results are obtained from McPAT.

The major factor that limits the HAAG$ size is the available empty space in the logic die left by the shared common peripheral circuitry (input buffer, rank controller, decoder, bus, and TSVs). In our 3D DRAM configuration, the logic die has 0.34mm^2 available for every vertical rank in the stack based on our area model. We fill this space with a 64KB SRAM-based HAAG$, which has an area of 0.27mm^2. With the page (i.e., row) size of 2KB from the best stacked DRAM configuration, the HAAG$ in each rank can hold a maximum of 32 DRAM rows (2KB each).

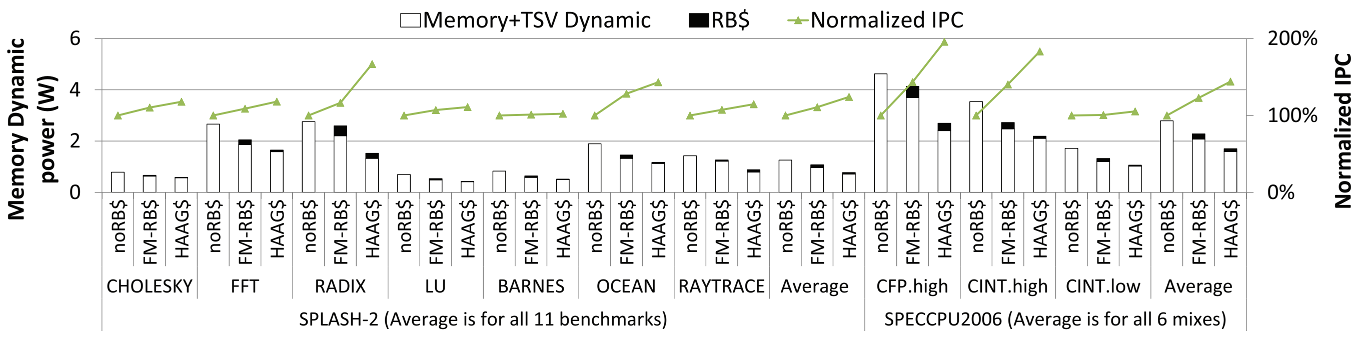
Also, the HAAG$ access granularity has an upper bound of the DRAM page size and a lower bound of 256B as discussed in Section 3.1. Accordingly, the associativity is flexible from 32 to 256. In addition, the HAAG$ has a data block size and IO width of 64B (a CPU cache line). We assume there is an address controller in the HAAG$ so that a cacheline address can be incremented to assemble multiple small entries for the entire or a partial page (row). Finally, the memory controller queue and the predictor queue are assumed to have 32 and 128 entries respectively.

The total processor-to-memory bandwidth of the system is determined by the number of memory controllers and the bandwidth of each hybrid memory stack connected to the memory controller. The selected design configuration shown in Table 2 can achieve 280GB/s internal bandwidth with high bank concurrency, and we assume an external bandwidth of 128GB/s between the processor and the memory stack (Figure 6). Both these bandwidths match the HBM interface standard definition [12] and the hybrid memory cube design target by Micron [29]. The 128GB/s memory bandwidth is distributed to two memory channels of a memory controller. All internal ranks share the external memory channels and communicate to the processor on the same silicon interposer. The memory controller uses parallelism-aware batch-scheduling (PAR-BS) [28] for scheduling memory accesses. Within a request batch, it prioritizes read requests over writes as reads are typically more latency sensitive.

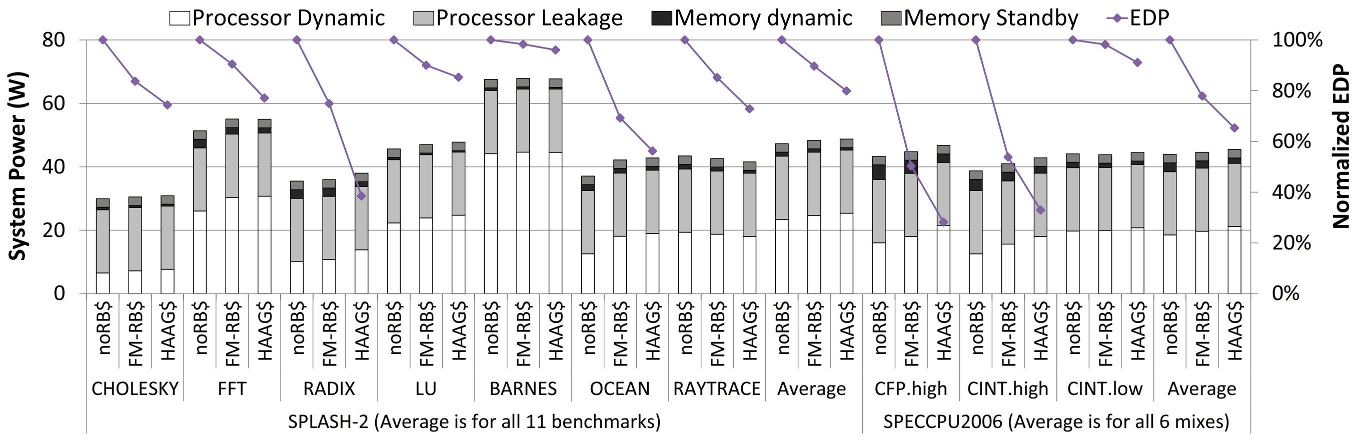
5.2 System Performance and Energy Efficiency

Based on all the timing and energy parameters of HAAG$, 3D DRAM, and processor obtained in Table 2, we demonstrate the improvement in performance and energy efficiency that HAAG$ brings into the system. Figure 7 demonstrates power, instructions per cycle (IPC), and system energy-delay product (EDP) of the baseline 3D DRAM architecture without row-buffer caches (RB$s), the architecture that simulates Loh's proposal (FM-RB$), and our proposed History-Assisted Adaptive-Granularity Cache (HAAG$). Because the HAAG$ explores the synergy of the inseparable history-assisted predictor in addition to the adaptive-granularity cache to achieve performance and power improvements, we evaluate the design's efficiency in its entirety.

Figure 7a shows the IPC and the dynamic power breakdown of the memory stack. As we can see, the HAAG$ memory subsystem is clearly better in terms of both performance and power consumption than the state-of-the-art FM-RB$. In particular, applications with high bandwidth demands such as RADIX, OCEAN, CFP.high, and CINT.high show up to 43.5% performance improvement (33.5% on harmonic average) and 41.3% power reduction (28.9% on average) compared to FM-RB$. Figure 7a shows normalized IPCs of all workloads so that they are at the same scale. For reference, the absolute IPC values examples of RADIX, OCEAN, CFP.high, and CINT.high running on a system with HAAG$ are 10.2, 13.1 15.2, and 15.5, respectively.



(a) Memory dynamic power and IPC (normalized against no-RB$ for each workload)



(b) Normalized EDP (normalized against no-RB$ for each workload)

Figure 7: Power, IPC (higher is better), and normalized EDP (lower is better) of the baseline, FM-RB$, and HAAG$ systems running the SPLASH-2 and SPEC CPU2006 benchmarks. 'noRB$', 'FM-RB$', and 'HAAG$' denote (single memory stack) system without row buffer cache (RB$), with 'FM-RB$', and with 'HAAG$', respectively. Note that the harmonic average numbers in the figures are over the entire benchmark suite of SPLASH-2 and SPEC CPU2006 respectively, although only a subset of benchmarks have their results shown in the figure due to limited space.

An important observation is that HAAG$ works particularly well for memory-intensive applications with irregular data access and limited locality (many memory-intensive multithreaded applications belong to this category), compared to FM-RB$. For instance, although RADIX is more memory-intensive than OCEAN, FM-RB$ performs better for OCEAN (28.2% over the baseline) than RADIX (16.3% over the baseline) due to the poor locality in RADIX. In comparison, HAAG$ performs better for RADIX (67.1% over the baseline) than OCEAN (43.2% over the baseline). Thus, HAAG$ improves performance regardless of access locality because of its ability to adapt to row-buffer cache granularity to match the behavior of applications. This is especially important for manycore CPU workloads where memory-side locality is hard to leverage [2].

Figure 7b shows normalized system EDP, where energy (E) includes dynamic and leakage parts for both the processor and the memory. The figure also illustrates the breakdown of total system power attributed to individual contributions from processor/memory and dynamic/leakage. HAAG$ has the best EDP for all the workloads with a maximum improvement of up to 48.7% and an average of 13.3% compared to FM-RB$. For memory-intensive benchmarks such as RADIX, OCEAN, CFP.high, and CINT.high, the EDP of HAAG$ improves more significantly (39.4% on harmonic average) compared to FM-RB$. This results from their higher IPC (i.e., better performance) and correspondingly much less application execution time and lower energy. Again, it proves that HAAG$ is much more effective in improving both performance and energy efficiency for memory intensive applications running on manycore systems by exploiting the hidden locality in memory accesses.

The improvement in performance and EDP results from the synergy of a higher row-buffer cache hit rate and reduced data transfers for smaller data granularity. Figure 8 shows HAAG$'s effectiveness in increasing cache hit rate. For the exemplified memory-intensive applications, the row-buffer hit rate (no-RB$) is as low as 15%-20% because the high thread count causes severe row-buffer conflicts. HAAG$ is able to increase the hit rate over the state-of-the-art FM-RB$ by an average of 24.1% for memory-intensive applications. The results in Figures 7 and 8 demonstrate the effectiveness of the adaptive-granularity scheme and the decoupled predictor.

5.3 Impact of System Memory Bandwidth

We then studied the sensitivity of HAAG$ to memory bandwidth on both in-order and out-of-order (OOO) processors by increasing the memory stack bandwidth. The results reveal the relationship of performance over bandwidth as shown in Figure 9. For OOO processors we employ 16 OOO Nehalem-like cores running at 3GHz, with a peak performance of 192 GOPS/s that matches the 64-core in-order processor. Each OOO core has a 64-entry reorder-buffer, a 128-entry instruction queue, 32KB 4-way set associative L1 I/D

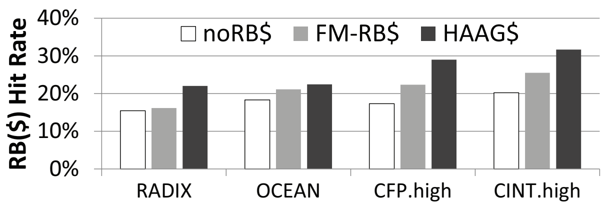


Figure 8: Comparison of row-buffer cache hit rates for RB$ designs with memory-intensive workloads (RADIX, OCEAN, CFP.high, and CINT.high). For no-RB$ case we show row-buffer hit rate.

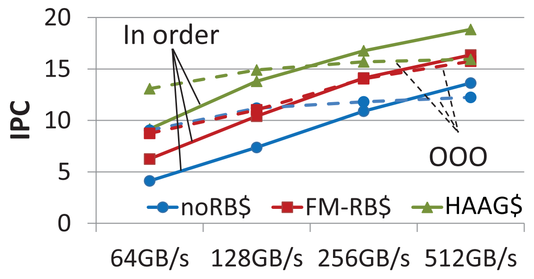


Figure 9: Comparison of IPC results averaged over memory-intensive workloads (RADIX, OCEAN, CFP.high, and CINT.high) for different external memory bandwidths on both in-order and out-of-order processors. The stack's internal to external bandwidth ratio is fixed at 2×.

caches and a 1MB 16-way set associative L2 cache. Although it has the same theoretical throughput as the in-order processor configuration, the OOO processor has better single thread performance but lower throughput than the in-order processor in practice, especially when provided sufficient memory bandwidth.

Benchmarks with relatively high memory bandwidth demands (i.e., RADIX, OCEAN, CFP.high, and CINT.high) see improvement in performance as we scale provisioned external system memory bandwidth from 64GB/s to 512GB/s for both in order and out-of-order processors. HAAG$ is more advantageous than both FM-RB$ and no-RB$ designs for all external system memory bandwidth configurations. Especially when the available system memory bandwidth is low, HAAG$ performs much better than the other two schemes. This is because with a limited available external system bandwidth, HAAG$ makes the best use of the internal memory bandwidth by caching data more judiciously.

When memory bandwidth increases, requests tend to be served faster with less contention at the memory controller. Thus, the relative performance differences among noRB$, FM-RB$, and HAAG$ become smaller. However, it is important to note that the performance of HAAG$ at a lower available external system memory bandwidth (e.g., 128GB/s) is similar to that of FM-RB$ at a higher bandwidth (e.g., 256GB/s), and that of no-RB$ at an even higher bandwidth (e.g., 512GB/s). This also further demonstrates that HAAG$ requires much less external system memory bandwidth than FM-RB$ for achieving the same performance, and thus saves cost and energy on memory and links.

6. CONCLUSION AND FUTURE WORK

3D stacked DRAM is a promising solution to the ever-increasing demands for memory capacity and bandwidth in supercomputers and datacenters. However, the limited in-stack data bus width becomes a bottleneck for fully unleashing performance and energy benefits from 3D DRAM. In this work we proposed the History-Assisted Adaptive-Granularity Cache (HAAG$) to best utilize the narrow TSV channel width in the 3D DRAM stack, and fully exploit the potential of 3D DRAM to maximize performance and energy efficiency at the system level. A HAAG$ employs an adaptive caching scheme that dynamically adjusts the amount of data to cache and an intelligent predictor that makes decisions on whether and how much data to cache from a DRAM row. With these two synergistic techniques, HAAG$ significantly improves memory access latency and memory system energy efficiency by increasing the row-buffer cache hit rate and avoiding unnecessary data transfers. For important memory-intensive SPLASH-2 and SPEC CPU-2006 workloads, HAAG$ outperforms the state-of-the-art by 33.5% on average. It is important to note that although evaluated in the context of 3D DRAM architecture, the proposal is not constrained to 3D DRAM and can be implemented in other memory systems with buffer-on-board (BoB) chips as well, such as the DDR3/4 interfaced load-reduced DIMMs (LRDIMMs) with LR-Buffers. We leave the application of HAAG$ to other memory systems as future work.

The goal of our proposed HAAG$ is making DRAM row-buffer cache designs efficient and practical so that their advantages can be well leveraged for maximizing performance and energy efficiency of memory systems in general. The innovations of adaptive granularity and history-assisted prediction work especially well for memory systems with a modest data bus width, cost-constrained cache size, and scalable memory capacity. Especially since future CPUs will get more and more cores and row locality will go down dramatically, HAAG$ with its adaptivity will become more important and will be a much needed innovation.

Acknowledgments

We thank the anonymous reviewers for their insightful comments to improve our paper. Ke Chen and Jay Brockman were partially supported by the C2S2 Focus Center, one of six research centers funded under the Focus Center Research Program (FCRP), a Semiconductor Research Corporation entity. Jung Ho Ahn was partially supported by the Future Semiconductor Device Technology Development Program (10044735) funded by MOTIE and KSRC. Cong Xu and Yuan Xie were supported in part by NSF 1500848, 1017277, and 1461698. This material was also partially based on work supported by the Department of Energy under Award Number DE - SC0005026.

7. REFERENCES

[1] "Hybrid Memory Cube Specification 1.0," Hybrid Memory Cube Consortium, Tech. Rep., Jan 2013.

[2] J. Ahn, N. P. Jouppi, C. Kozyrakis, J. Leverich, and R. S. Schreiber, "Future Scaling of Processor-Memory Interfaces," in *Supercomputing*, 2009.

[3] J. Ahn, S. Li, S. O, and N. P. Jouppi, "McSimA+: A Manycore Simulator with Application-level+ Simulation and Detailed Microarchitecture Modeling," in *ISPASS*, 2013.

[4] R. Ausavarungnirun, K. K.-W. Chang, L. Subramanian, G. H. Loh, and O. Mutlu, "Staged Memory Scheduling: Achieving High Performance and Scalability in Heterogeneous Systems," in *ISCA*, 2012.

[5] K. Chen, S. Li, N. Muralimanohar, J. B. Brockman, and N. P. Jouppi, "CACTI-3DD: Architecture-level Modeling for 3D Die-stacked DRAM Main Memory," in *DATE*, 2012.

[6] X. Dong, Y. Xie, N. Muralimanohar, N. Jouppi, and R. Kaufmann, "Simple but Effective Heterogeneous Main

Memory with On-chip Memory Controller Support," in *SC*, 2010.

[7] X. Dong and Y. Xie, "System-Level Cost Analysis and Design Exploration for Three-Dimensional Integrated Circuits (3D ICs)," in *ASP-DAC*, 2009.

[8] T. J. Ham, B. Chelepalli, N. Xue, and B. Lee, "Disintegrated Control for Energy-Efficient and Heterogeneous Memory Systems," in *HPCA*, 2013.

[9] H. Hidaka, Y. Matsuda, M. Asakura, and K. Fujishima, "The cache DRAM Architecture: A DRAM with an On-Chip Cache Memory," vol. 10, no. 2, pp. 14–25, 1990.

[10] IC Konwledge LLC, "IC Cost Model Revision 1105 http://www.icknowledge.com/," 2011.

[11] K. Inoue, K. Kai, and K. Murakami, "Dynamically Variable Line-Size Cache Exploiting High On-Chip Memory Bandwidth of Merged DRAM/Logic LSIs," in *HPCA*, 1999, pp. 218–222.

[12] JEDEC, "High Bandwidth Memory (HBM) DRAM - JESD235," Oct 2013. [Online]. Available: http://www.jedec.org/standards-documents/docs/jesd235

[13] D. Jevdjic, G. H. Loh, C. Kaynak, and B. Falsafi, "Unison Cache: A Scalable and Effective Die-Stacked DRAM Cache," in *MICRO*, 2014.

[14] U. Kang *et al.*, "8 Gb 3-D DDR3 DRAM Using Through-Silicon-Via Technology," *JSSC*, vol. 45, no. 1, pp. 111–119, 2010.

[15] G. Kedem and R. P. Koganti, "WCDRAM: A Fully Associative Integrated Cached-DRAM with Wide Cache Lines," Duke University, Tech. Rep., 1997.

[16] T. Kgil, S. D'Souza, A. Saidi, N. Binkert, R. Dreslinski, T. Mudge, S. Reinhardt, and K. Flautner, "PicoServer: Using 3D Stacking Technology to Enable a Compact Energy Efficient Chip Multiprocessor," in *ASPLOS*, 2006.

[17] J.-S. Kim *et al.*, "A 1.2V 12.8GB/s 2Gb Mobile Wide-I/O DRAM with 4x128 I/Os Using TSV-Based Stacking," in *ISSCC*, 2011.

[18] P. Kongetira, K. Aingaran, and K. Olukotun, "Niagara: A 32-Way Multithreaded Sparc Processor," *IEEE Micro*, vol. 25, no. 2, pp. 21–29, 2005.

[19] D. Lee, Y. Kim, V. Seshadri, J. Liu, L. Subramanian, and O. Mutlu, "Tiered-latency DRAM: A Low Latency and Low Cost DRAM Architecture," in *HPCA*, 2013.

[20] S. Li, J. 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 *MICRO*, 2009.

[21] S. Li, D. H. Yoon, K. Chen, J. Zhao, J. Ahn, J. B. Brockman, Y. Xie, and N. P. Jouppi, "MAGE: Adaptive Granularity and ECC for Resilient and Power Efficient Memory Systems," in *SC*, 2012.

[22] G. H. Loh, "3D-Stacked Memory Architectures for Multi-Core Processors," in *ISCA*, 2008.

[23] ——, "A Register-file Approach for Row Buffer Caches in Die-stacked DRAMs," in *MICRO*, 2011.

[24] G. Loi, B. Agrawal, N. Srivastava, S. Lin, T. Sherwood, and K. Banerjee, "A Thermally-Aware Performance Analysis of Vertically Integrated (3-D) Processor-Memory Hierarchy," in *DAC*, 2006.

[25] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, "Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation," in *PLDI*, Jun 2005.

[26] H. McGhan, "SPEC CPU2006 Benchmark Suite," in *Microprocessor Report*, Oct 2006.

[27] S. W. Moore and B. T. Graham, "Tagged Up/Down Sorter - a Hardware Priority Queue," *The Computer Journal*, vol. 38, pp. 695–703, 1995.

[28] O. Mutlu and T. Moscibroda, "Parallelism-Aware Batch Scheduling: Enhancing both Performance and Fairness of Shared DRAM Systems," in *ISCA*, 2008.

[29] J. T. Pawlowski, "Hybrid Memory Cube (HMC)," in *Hot Chips*, 2011.

[30] Samsung Electronics, "DDR3 SDRAM Datasheet," 2002.

[31] Semiconductor Industries Association , "International Technology Roadmap for Semiconductors." 2007.

[32] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically Characterizing Large Scale Program Behavior," in *ASPLOS*, 2002.

[33] Y. H. Son, S. O, H. Yang, D. Jung, J. Ahn, J. Kim, J. Kim, and J. W. Lee, "Microbank: Architecting Through-silicon Interposer-based Main Memory Systems," in *SC*, 2014.

[34] D. H. Woo, N. H. Seong, D. L. Lewis, and H.-H. S. Lee, "An Optimized 3D-Stacked Memory Architecture by Exploiting Excessive, High-Density TSV Bandwidth," in *HPCA*, 2010.

[35] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 programs: Characterization and Methodological Considerations," in *ISCA*, 1995.

[36] D. H. Yoon, M. K. Jeong, and M. Erez, "Adaptive Granularity Memory Systems: A Tradeoff between Storage Efficiency and Throughput," in *ISCA*, June 2011.

[37] Z. Zhang, Z. Zhu, and X. Zhang, "Cached DRAM for ILP Processor Memory Access Latency Reduction," in *IEEE Micro*, vol. 21, no. 4, 2001, pp. 22–32.