2017 IEEE International Symposium on High Performance Computer Architecture

SILC-FM: Subblocked InterLeaved Cache-Like Flat Memory Organization

Jee Ho Ryoo
The University of Texas at Austin
Austin, TX
jr45842@utexas.edu

Mitesh R. Meswani
*ARM**
Austin, TX
mitesh.meswani@gmail.com

Andreas Prodromou
UCSD
San Diego, CA
aprodrom@eng.ucsd.edu

Lizy K. John
The University of Texas at Austin
Austin, TX
ljohn@ece.utexas.edu

***Abstract*—With current DRAM technology reaching its limit, emerging heterogeneous memory systems have become attractive to continue scaling memory performance. This paper argues for using a small, fast memory closer to the processor as part of a flat address space where the memory system is composed of two or more memory types. OS-transparent management of such memory has been proposed in prior works such as CAMEO and Part of Memory (PoM). Data migration is typically handled either at coarse granularity with high bandwidth overheads (as in PoM) or at fine granularity with low hit rate (as in CAMEO). Prior work uses restricted address mapping from only congruence groups in order to simplify the mapping. At any time, only one page (block) from a congruence group is resident in the fast memory.**

In this paper, we present a flat address space organization called SILC-FM that uses large granularity but allows subblocks from two pages to coexist in an interleaved fashion in fast memory. Data movement is done at subblocked granularity, avoiding fetching of useless subblocks and consuming less bandwidth compared to migrating the entire large block. SILC-FM can achieve more spatial locality hits than CAMEO and PoM due to page-level operation and interleaving blocks respectively. The interleaved subblock placement improves performance by 55% on average over a static placement scheme without data migration. We also selectively lock hot blocks to prevent them from being involved in hardware swapping operations. Additional features such as locking, associativity and bandwidth balancing improve performance by 11%, 8%, and 8% respectively, resulting in a total of 82% performance improvement over a no migration static placement scheme. Compared to the best state-of-the-art scheme, SILC-FM gets performance improvement of 36% with 13% energy savings.

***Keywords*-die-stacked DRMA; subblocked; flat memory;**

I. Introduction

In the current era, where the bandwidth requirement of multi-core processors has exceeded the maximum bandwidth provided by conventional DDR DRAM technology, alternative memory technologies rise as a solution to leap over the bandwidth wall [1], [2]. Near term solutions are likely to combine alternative memory and conventional DRAM technologies to form a heterogeneous memory system. A few such popular technologies are Phase Change Memory (PCM), which provides a higher capacity, and die-stacked DRAM, which provides high bandwidth. These alternatives all play an important role to supplement the existing memory technology, yet integrating these technologies in existing systems remains challenging.

Emerging technologies such as High Bandwidth Memory (HBM) and Hybrid Memory Cube (HMC) offer much higher bandwidth than the conventional DDR technology [3], [4], [5], [6]. In addition, its physical proximity to the processor offers slightly reduced access latency. Over the years, the capacity of these modules has increased steadily from a few hundred megabytes to a few gigabytes, and exposing this capacity to users can make an impact for capacity constrained applications [7]. Although the capacity is much greater than conventional SRAM caches, many emerging applications often have a memory footprint of hundreds of gigabytes [8], [9], [10]. Therefore, even emerging memory technologies cannot hold all hot pages of such applications. Researchers have been exploring ideas to efficiently use this small, fast memory as a large Last Level Cache (LLC). Researchers focus on ways to manage tags (metadata) efficiently by managing data at different cacheline sizes to reduce the metadata storage overheads, while at the same time, they try to perform selective cacheline fetching from the off-chip DRAM to reduce off-chip bandwidth usage [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22].

While such memory technologies cannot replace the main memory yet, it still has significant capacity, which can contribute towards useful main memory capacity. The operating system (OS) can explicitly manage such capacity via page placement [23], [24], [25], [26], [27]. The OS monitors page usage and swaps hot pages from slow memory to fast memory at some fixed time intervals or epochs. The major drawbacks are the high software related overheads. The overheads to perform a bulk migration are significant, and hence the OS management is done at coarse granularity (hundreds of milliseconds). As a result, software schemes are typically slow to to react to changes to the hot working set.

To address this issue, recent proposals such as CAMEO and PoM schemes [7], [28] expose the fast memory capacity while transparently migrating data between off-chip and on-chip memories without involving the OS. CAMEO migrates 64B block at a time whereas PoM does so at 2KB. The migration is done using hardware remapping tables, so some off-chip memory data are remapped to die-stacked DRAM.

*This work was done while the author was working at AMD Research

2378-203X/17 $31.00 © 2017 IEEE
DOI 10.1109/HPCA.2017.20

In this work, we present Subblocked InterLeaved Cache-Like Flat Memory Organzation (SILC-FM), which fully utilizes the die-stacked DRAM capacity while intelligently placing hot pages in fast memory. The subblock level data movement reduces the bandwidth usage compared to PoM and prevents fetching data that may not get utilized much. However, SILC-FM fetches multiple small-size blocks at a time, resulting in more spatial locality hits compared to CAMEO. In effect, SILC-FM achieves the best of both CAMEO and PoM. Furthermore, SILC-FM allows interleaving between die-stacked and off-chip DRAM hot blocks, so subblocks from on-chip and off-chip DRAM can coexist together in faster memory. Our scheme is a hardware data management mechanism that is robust to common problems associated with recent proposals since SILC-FM can exclude and lock hot pages from being involved in undesirable data migration operations. Our associative structure also protects those pages that are not locked and are actively participating in hardware data migrations from being frequently swapped out. In addition, we maximize the memory bandwidth by utilizing both on-chip and off-chip memory bandwidth. We achieve this by moving only a certain subset of memory traffic to off-chip memory rather than servicing all memory requests from fast memory. Thus, we further improve performance by utilizing the sum of both on-chip and off-chip memory bandwidth. Our technique provides the following insights:

- The high bandwidth costs typically associated with page-level data placement and migration in prior PoM can be eliminated by using subblocking. **Subblocking** together while **interleaving** subblocks from two different pages increase the usefulness of the fast memory layer.
- Hot pages are preferred to reside in on-chip memory as those pages benefit from high bandwidth. We developed a mechanism to identify hot pages and to **lock** them in fast memory, so that conflicting data does not swap hot pages out to off-chip DRAM. Unlike epoch based approaches, the locking does not need to occur at coarse grain time intervals, and thus our scheme can react quickly to changes in the hot working set.
- In memory bandwidth limited systems, the bandwidth is the scarce resource. Thus, instead of maximizing the total number of requests serviced from the fast memory layer, it is beneficial to service a fraction of requests from the off-chip DRAM to utilize the overall available system bandwidth. If the bandwidth available from the two memory levels are N+1, it is beneficial to service 1/(N+1) of the accesses from the slower memory layer. Therefore, our scheme achieves high performance improvement with a slightly lower number of requests serviced from die-stacked DRAM. This leads to well **balanced** bandwidth utilization, which in turn improves performance.

We evaluate our scheme against the state-of-the-art scheme, which also fully utilizes the added fast memory capacity as a part of memory. Our scheme outperforms the state-of-the-art scheme by 36%. Also, with die-stacked DRAM's low energy consumption, our scheme reduces the Energy-Delay Product (EDP) by 13%.

II. Background

In this section, we first discuss challenges associated with architecting small, fast memory as a part of memory. In this paper, our main focus is to efficiently use this memory as a part of memory, so we explain using prior state-of-the-art schemes. We show inherent advantages/disadvantages associated with each scheme. Throughout the paper, we refer the small, fast memory as Near Memory (NM) and off-chip DRAM as Far Memory (FM) since NM is physically located closer to the cores with high bandwidth. Also, we call 64B worth of contiguous address space as a small block or subblock and 2KB worth of contiguous address space as a large block. For software related schemes, we assume that the page size is 2KB.

A. Challenges with Architecting Near Memory as Part of Memory

Using emerging memories as part of memory rather than caches poses sophisticated design challenges. First, when swapping is done without OS intervention, a remap table is necessary to allow dynamic migration of data between NM and FM. However, keeping an SRAM based remap table for every small block will easily exceed today's on-chip SRAM capacity [7]. As NM scales to a larger capacity, the remap table storage overheads become a problem. Therefore, efficiently storing such large metadata is challenging and important. Making the remapping granularity to a large block (2KB) can mitigate this issue, yet on each remapping migration, the bandwidth usage becomes exorbitant as unnecessary subblocks also have to be migrated. Furthermore, recent work [15] has shown that as NM capacity scales, even making the remap granularity larger becomes problematic again. Using the OS to intervene the system and perform the remapping can completely eliminate such remapping overheads. Yet, this has to be done at a large time interval to minimize the OS related overheads, so its adaptation to memory behavior changes is inherently slow. Therefore, carefully choosing the remap granularity and the frequency is crucial. Similarly, identifying what to placed in NM is an important task since only a subset of the entire memory space can be held in NM. Current OS main memory activity monitoring is done at page granularity based on the reference bit in a Page Table Entry (PTE). This method limits the OS to accurately select the appropriate number of hot blocks that will fit in NM [25]. Therefore, a finer monitoring method is necessary.

B. Hardware Schemes

A block based scheme swaps either small or large size blocks between two memories upon a request. This category of schemes attempts to exploit temporal locality as they expect those swapped blocks in NM to be used frequently. Each block in NM has a remap table entry, which identifies whether a requested block resides in NM or FM. A scheme can use different data management granularities from a small block (64B) to a large block (2KB). The key idea in this category of schemes is that while it allows NM capacity to be exposed to the OS, the dynamic remapping between NM and FM allows the data to move between two memories without OS intervention at low overheads. PoM [28] and CAMEO [7] are two state-of-the-art schemes that use a large and small block size respectively. PoM's large blocks are migrated to NM intelligently by estimating the benefit and cost of migrating pages. If the benefit outweighs the cost, then it is migrated to NM. PoM requires a counter for a page to reach a threshold before migration occurs, and thus it misses potential opportunities. Also, although only a subset of 2KB page is desired, it has to fetch the entire 2KB, which wastes significant bandwidth in low spatial locality workloads. Unlike PoM, CAMEO adopts a small block size. Since the migration bandwidth consumption is low, it allows small blocks to swap from FM whenever a requested small block is in FM.

CAMEO manages data at a small block granularity, so each block must have an accompanying remap table entry. The high metadata storage overheads due to having a remap table with each block is a drawback. Therefore, the metadata, namely the remap table, is stored next to data within the same row in the large NM. During an NM access, the burst length is increased to fetch the extra bytes of metadata, and this saves latency as only one memory request is required per access instead of two. Prior work [20] showed the difficulties involved with associative structures in NM as fetching multiple data in parallel from the same NM row is not possible. Consequently, the direct-mapped approach is preferred to achieve low latency. Thus, conflict misses are an inherent problem in this scheme. Also, by only swapping a small block at a time from FM, this scheme does not take advantage of the abundant spatial locality present at a large block level. The original CAMEO paper does not implement any prefetching scheme, which might benefit high spatial locality workloads, thus it achieves a lower hit rate. Furthermore, other work [15], [16] has shown CAMEO's lost performance opportunities by only fetching 64B at a time. Therefore, in this paper, in addition to original CAMEO, we have additionally also evaluated CAEMO with prefetching to see higher spatial locality effects.

C. Software Schemes

In an epoch based scheme, the OS explicitly manages the NM capacity as a special region of memory. The OS moves hot pages (2KB) into NM and the PTE is updated accordingly. Upon an access, the post-translation physical address is directly used to access data in NM. Unlike hardware managed schemes, it does not have additional hardware structures like the remap table or modified addressing schemes, which result in unconventional data layout in a row. In CAMEO, the addressing scheme in the memory controllers has to be modified to fetch the remap table entry located next to every data block. Yet, the NM data layout and addressing of epoch based schemes is the same as in conventional DRAM, so it does not require specialized logic in memory controllers. The key to achieve high performance in epoch based schemes is the ability to identify hot pages and to reduce migration related overheads. In this section, we use the state-of-the-art epoch based scheme, the HMA scheme [25], to describe the advantages and disadvantages.

The software scheme relies on hot page detection to achieve performance improvement as there is no additional hardware to perform sophisticated operations such as dynamic remapping. HMA uses a dynamic threshold based counter where the pages, whose access counts are higher than a set threshold, are marked using an unused bit in the PTE. The page migration occurs at a large interval, and the interval is referred to as an epoch. At each epoch boundary, the OS sweeps through the PTEs to select those pages that are marked, and the bulk page migration occurs between NM and FM. In addition to the time spent on physically transferring pages, this operation requires the system to update PTEs and invalidate corresponding TLB entries. Costs related to such operations are extremely high, and thus the benefit of performing the block migration has to be large enough to offset the expensive costs [25], [28].

This scheme is slow to changes in the hot working set, which is a common behavior in many applications with different execution phases. For example, a page can become hot in FM, yet this page cannot be serviced from NM until the next epoch, which may take millions of cycles to reach. Until then, the potential to get performance benefits when it is placed in NM is not exercised. Furthermore, the working set coverage by NM is fixed during an epoch as no data migration occurs between NM and FM except at epoch boundaries. If a larger working set coverage is desired, the hardware management scheme can swap blocks from FM and a larger amount of data can be served from NM at the end of an epoch.

III. SILC-FM Memory Architecture

In this section, we will explain the details of the SILC-FM architecture piece-by-piece. The SILC-FM scheme uses NM as OS visible space while internally operating with subblock based mechanisms. NM is organized as an associative structure where subblocks are swapped between NM and FM. A locking feature prevents hot blocks from being swapped out to FM while a bypassing feature utilizes the overall

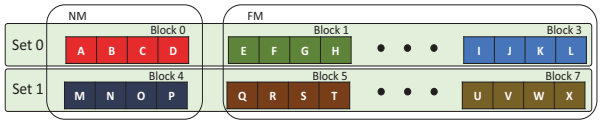

Figure 1: NM Set Mapping

system bandwidth. Figure 1 shows the initial memory state where each row is a congruence set. In this example, each block contains four subblocks and the mapping is direct-mapped. This implies that multiple subblocks from only one large block (from the same set) in FM can swap into corresponding subblocks in NM at any one point of time. For example, subblock A and C are in the same large NM block whereas another subblock F is in a different large block in FM, so the subblocks F and J can only swap into subblock B in NM. At any point in time, only one of subblock F or J can reside in NM. In Figure 1, the migration between two pages within the same set occurs at subblock granularity. This is bandwidth efficient as only 64B worth of data is migrated, yet managing the remap table at a large block granularity reduces the remap table overheads. The subblock tracking is done using a bit vector per NM block where individual bits validate the corresponding subblock's residency in NM. When a block is swapped out of NM, the history of bit vectors is stored in a small SRAM structure called a bit vector table. Our bit vector history table has a total of 1 million entries (˜4MB). Multiple subblocks are fetched using this bit vector when a block is swapped in again. This exploits spatial locality as previously used subblocks are swapped at the same time, so any subsequent request to either of the subblocks results in a subblock serviced from NM. In comparison to CAMEO, our scheme can achieve higher spatial hits. Since this scheme does not swap any other undesirable subblocks, it is more bandwidth efficient than large block based schemes such as PoM, which have to swap every subblock within a large block.

Since NM is not a cache, we do not use the term "hit rate" to describe the fraction of requests serviced from NM. Rather, we adopt the term, "access rate", that is used in a recent die-stacked DRAM paper [7]. The access rate is defined as

$$AccessRate = \frac{\frac{total\ number\ of\ requests}{serviced\ from\ NM}}{\frac{total\ number\ of\ requests}{missed\ from\ LLC}}. \quad (1)$$

In this work, we assume that NM uses the lower addresses in the physical address space and FM uses the higher addresses.

A. Hardware Swap Operations

Since NM is a part of memory space, when data is brought into NM, the existing data from NM needs to be swapped out. Unlike hardware caching schemes where there is always a copy in FM, SILC-FM needs to perform a swap operation. Figure 2 shows a direct mapped scheme to describe the swapping operation for subblocks F and H. When two back to back requests are made to subblock F and H, they are

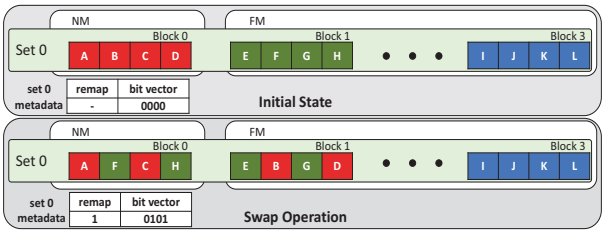

Figure 2: Example of Interleaved Swap

brought in one by one from block 1 in FM into block 0 in NM. The corresponding subblocks (subblock B and D) in block 0 are swapped out to block 1. Any subsequent access to subblock F and H will be serviced from NM. There are no duplicate copies of data and hence the total memory capacity is the sum of NM and FM capacities, which is much greater than in cache schemes such as Alloy Cache [20]. In SILC-FM, the address is used to calculate an index, which refers to a set of unique NM blocks. Upon a memory request, the index is calculated by performing the modulo operation with the incoming address and the total number of blocks in NM. Then, the incoming address is checked against the remap field of the metadata. The remap entry is only used for the swapped in data from FM since it contains the large block address of the swapped in block.

Subblocking is done via a bit vector, which consists of valid bits to indicate whether a particular subblock in this NM block has been swapped in from FM. In Figure 2, when subblock H is brought into NM, the corresponding bit, which is calculated using the block offset, is set in the bit vector. Also, the remap table entry is updated, so it contains the block address of the swapped in block, which is 1 in our case. Note that SILC-FM does not have a valid bit at block granularity because unlike caches, there is always data in NM, so the block is always valid. The difficulty here is to distinguish which block the data belongs to. SILC-FM can achieve that by using the NM address range and remap entry. Using the remap table entry, bit vector and the request address, there can be 6 scenarios of swap operations. Table I lists each operation and we will refer to this table to explain each swap operation. First, we begin with the case where the remap entry matches with the request address. If a bit in the bit vector is not set and the request address belongs to the NM address space, then the subblock (original NM subblock) is resident. Otherwise, if the bit is set, then the original subblock is swapped out to FM.

In the scenario with a remap entry mismatch, the bit is

Remap	Bit Vector	NM Address	Action
match	1	-	service from NM
match	0	-	swap subblock from FM
mismatch	1	yes	swap subblock from FM
mismatch	0	yes	service from NM
mismatch	1	no	restore current block and swap subblock from FM
mismatch	0	no	restore current block and swap subblock from FM

Table I: Metadata and Operation Summary

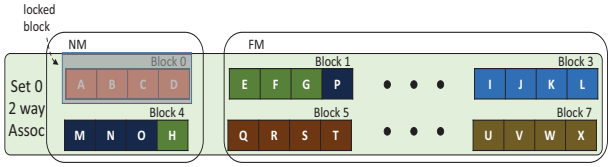

Figure 3: Locking and Associativity

set, and the address falls in the NM address space, then the swapped out subblock (the one originally belonging to NM) is brought into NM. The FM location where the swapped out subblock currently resides is the block address in the remap entry, so the remap entry is consulted to bring the subblock back. In the same scenario with the bit not set, the original subblock is resident, so the request is serviced from NM. Lastly, if the remap entry mismatches and the requesting block address belongs to the FM space, the original mapping is restored. Although the bit vector is not used to make an execution action, the bit vector is consulted to restore the original mapping. At the same time, the bit vector is saved in a bit vector table. This is a small SRAM structure that is indexed using the xor'ed PC and address of the first swapped in subblock within the block (first in timely sense). Thus, this PC and request address is stored along with other metadata for each block. These two variables have shown to have high correlation with the program execution [29], [30], [31], [32], so when this block is accessed again in the future, it is likely that a similar access pattern will repeat. Since the bit vector has a pattern of previous subblock usage, it will be used to fetch multiple subblocks when this block is swapped into NM again. The bit vector table is indexed and those subblocks corresponding to 1's in the bit vector are fetched together from FM, which takes advantage of spatial locality. By doing so, SILC-FM can achieve a higher access rate than small block schemes. Now, after successfully restoring a block, the new swapping occurs between NM and FM.

If a FM subblock has to be swapped into NM, the corresponding subblock in NM (the subblock that originally belongs to the NM address space) is swapped out to FM. The corresponding bit is set accordingly and the subblock is now resident. Since our large block size is 2KB and the subblock size is 64B, there are 32 bits per block. Each bit is responsible for each subblock position in NM. Unlike caches, we do not keep track of dirty bits in NM. In caches, dirty bits facilitate the eviction process by only writing back dirty data and invalidating clean data. However, since data in NM is the only copy of the data in the physical address space, all swapped in blocks need to be written back to FM when necessary. Therefore, our scheme does not need dirty bits.

B. Memory Activity Monitoring

SILC-FM monitors memory access activities to classify data into hot and cold data. The idea is that we want to keep only hot pages in NM to benefit from NM's high bandwidth. The cold data should not interfere with hot data to keep hot data from being inadvertently swapped out to FM. The activity tracking metadata, namely NM and FM counters, are used to gather memory access statistics, and each page in NM has its own dedicated set of NM and FM counters. Each NM page has two counters, each with 6 bits, so the total area overhead is 1.5MB, which is negligible.If there are any swapped in subblocks in NM, then two different blocks coexist in the same NM row; ones originally in NM space and the others swapped in from FM. Unlike the remap entry where only one entry is needed to distinguish those two sets, for monitoring activities, we need two sets of counters, each for NM and FM blocks. The counter is used to classify two coexisting blocks into either hot or cold blocks. Upon an access, the updated counter value is compared against a threshold and if it exceeds this value, the large block is hot. Otherwise, it is considered cold. This later helps to identify candidates to lock in NM. In order to distinguish between current and past hot blocks, these counters are implemented using aging counters where the counter value is shifted to the right at every one million memory accesses.

C. Locking Pages

Once the system identifies hot blocks, SILC-FM locks hot large blocks in NM as those blocks are responsible for high bandwidth usage, which benefits from being placed in NM. When the counter crosses the threshold, the block is locked. To reduce hardware complexity, the locking is done at large block/page granularity although the unlocked pages still operate at subblock granularity. Unlike unlocked blocks, the locked blocks have all their subblocks in NM. Therefore, when locking the block, the missing subblocks, which are residing in FM, are swapped into NM. After locking, a complete large block remap has been performed as the large block originally belonging to NM is now completely remapped to a location in FM and vice versa. The counter for this locked page is still incremented upon each access, but the bit vector check is ignored. The counter is still monitored to ensure that the locked block is still hot. If the locked block is no longer hot and the access count becomes below the threshold, the lock bit is unset. Clearing the lock bit does not have an immediate effect as it operates as if the unlocked block has all subblocks swapped in from FM (all bits in the valid bit vector are set). If this block has, in fact, lost its hotness, then other hot subblocks will be swapped into this place in NM. Our locking and unlocking mechanism can react more quickly to changes in the hot working set than epoch based schemes as migrating hot blocks do not have to wait until epoch boundaries. In fact, our scheme does not have a notion of epochs, so locking and unlocking can happen at anytime for any number of blocks.

However, locking a block makes other subblocks in the same set in FM inaccessible to NM as they can only be swapped into NM in the same set when a direct-mapped scheme is used. Therefore, SILC-FM allows swapped in

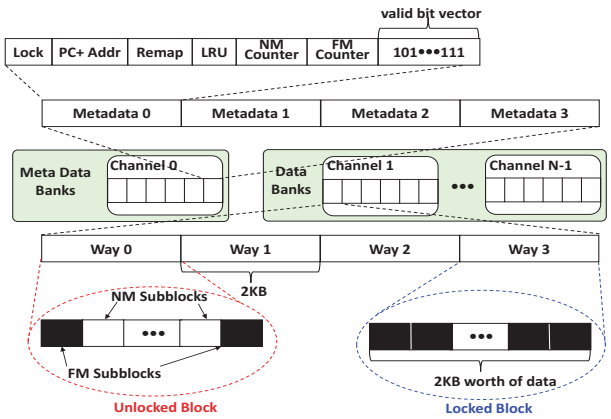

Figure 4: SILC-FM Data and Metadata Organization

subblocks to be placed with some flexibility by allowing block level associativity. We have experimented with various associativities at a management granularity of a large block (2KB). 1-way is a direct-mapped organization and it achieves the least performance improvement as several hot blocks get swapped out due to conflicts and thrashing. Increasing it to two improves performance by removing many conflicts. Yet, the index is still calculated using a part of the address bits, so multiple hot blocks (more than two) are still mapped to the same set. Consequently, increasing the associativity to four further improves the performance. As a result, SILC-FM adopts a four-way associative structure. Therefore, the incoming address is checked again all ways to find the matching remap entry. Prior page based work [28], [15], [13] showed similar results regarding associativity, yet our associativity is distinct from prior work in that depending on the number of pages locked, the associativity can vary from 1-way to 4-way.

D. Metadata Organization

The overall metadata organization in SILC-FM is shown in Figure 4. The block address for those swapped in subblocks are stored in the remap field as shown in the figure. The FM counter tracks the number of accesses made to these swapped in blocks. The FM counter is used since those subblocks originally belong to the FM address space. The block corresponding to Way 0 shows the layout of the data where subblocks shaded in black within 2KB block represent swapped in subblocks. Those swapped in subblocks originally belong to FM address space, but they are brought into NM upon a request. The white subblocks are data belonging to the original page that resides in NM address space. Again, since NM is an OS visible space just like FM, it originally has its own exclusive data from FM. Upon a request, this remap entry has to be checked to determine if the requested block has been swapped in or not. SILC-FM maintains one remap entry per large block/page while the residency of subblocks within a block is validated using a bit vector. SILC-FM also has other metadata fields such as lock and LRU, which are used for locked pages and finding the swapping candidate in NM. The metadata is stored in a separate channel to increase the NM row buffer hit rate of accessing metadata. Separating the metadata storage from data has been shown to increase the row buffer locality [13].

E. Bypassing and Bandwidth Balancing

Always swapping subblocks upon access increases the overall access rate. Although such approach makes sense for caches where NM is considered another level in memory hierarchy, it may leave the available bandwidth to FM idle once the access rate becomes high. In caches, FM being idle is actually beneficial since all requests are serviced from a layer of memory closer to the processor. Yet, in our situation where NM is a part of the memory space just like FM, having a portion of the memory being idle has a similar effect as disabling a fraction of the memory in the same memory hierarchy. This is not a desirable outcome, and in this case, making use of the FM bandwidth can increase the effective available bandwidth to the system. We have experimented with the effects of bypassing using the CAMEO scheme and steered an appropriate amount of traffic to match our desired access rate. Unlike SRAM caches where the maximum performance is achieved at 100% hit rate, in our experiment, the optimal performance improvement point in our case is at 0.8 instead of 1.0. In our system, the available bandwidth ratio between FM and NM is 4:1, so routing 80% of the traffic to NM and 20% to FM matches this bandwidth ratio. Since NM and FM are at the same memory hierarchy, using this FM bandwidth allows the processor utilize all bandwidths available in the system at the NM and FM layer. Thus, it is able to gain extra performance from the memory system. Since our work focuses on memory bandwidth bottlenecked system, having more bandwidth available to the application helps. Prior work [33], [34] also showed that bandwidth throttling is effective in bandwidth constrained systems, yet our bypassing is at much finer granularity (at page level) than in prior work, which uses a segment of memory as bypassing zones. In SILC-FM, we want to match the access rate to be 0.8, so we add the bypassing feature if the access rate exceeds this value. When this happens, no more subblocks are swapped into NM. However, the unlocked blocks, which are already in NM, can still operate normally from NM. For example, subblock G in Figure 3 can be swapped into block 1 upon a request and still work under unlocked conditions. However, in the bypassing scenario, this swapping is not allowed, so the bit vector will not be updated and subblock G will be serviced from FM. However, if the access rate again becomes lower than 0.8, this bypassing feature is turned off to increase the access rate.

F. Latency Optimization

Having an associativity structure adds the NM access latency as fetching multiple remap entries is a serialized operation. In SRAM caches, it does not add a significant latency as multiple entries can be fetched and checked simultaneously. However, it is not the case for NM since it uses DRAM based technology. Our implementation uses 4 way associativity, so in each access, four independent remap entries are fetched. This operation has to be serialized as the maximum fetch bandwidth is limited by the bus width. The metadata serialization problem was also addressed in prior work [18], [20]. In order to hide the long latency of fetching multiple remap entires, we add a small predictor that can bypass this serialization. The predictor has 4K entries in total. The predictor uses the instruction and data addresses since they are known to have a strong correlation with the execution phase of a program [29], [30], [31], [32], and thus, they are widely used as predictors in DRAM caches [16], [13], [20]. The program counter and data address offset values are xor'ed to form an index into the predictor. This table keeps track of a recently accessed way for each particular index. On each access, the index is calculated to access this table. Since this table is a small structure, the access latency is negligible. However, we assume that the access to the table begins with an LLC access, so by the time the LLC miss is identified, the predicted way is available to access NM. Furthermore, we add one more bit in each entry to speculate on the location of the data (NM or FM).

The latency on an access to FM is longer than an access to NM since the remap entry has to be checked first in NM prior to accessing FM. Our predictor attempts to improve this issue. If the predictor speculates that the data is located in FM, then the request is sent to FM at the same time as the remap entry request is sent to NM. Upon correct speculation of the data in FM, the latency is just a single FM access latency, hiding the NM remap entry fetching latency. Therefore, the saved time is the NM access latency. Note that the predictor only forwards the requests to FM when the block is speculated to exist in FM. In the case where the block is speculated to be in NM, no additional action is taken. If this prediction was not correct (FM speculated, but data is in NM), the simultaneously forwarded request to FM is ignored.

G. Putting It All Together

Now, the overall scheme is explained from the point of an LLC miss using Figure 5. The congruence set index is calculated using the modulo operator to access both the remap entry and data in NM. Also along with LLC access, the PC and the request address are used to access the predictor. The request is sent to NM using the calculated index and predicted way. If the lock bit and the remap field is set and matched, the NM data is fetched. In case of a

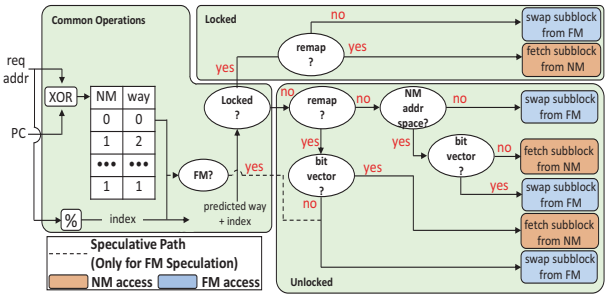

Figure 5: Overall Execution Flow of SILC-FM

remap mismatch, the request address is checked whether it falls under the NM address space. If so, the bit vector is checked to determine the location of the requested subblock (if resident in NM, the bit has to be 0). The prediction, shown in a dotted line, can skip the previously mentioned metadata fetching steps. If the block address does not fall under NM space, then the remap entry update and subblock swap from FM are initiated. The subblock is swapped to available ways within the set. A similar operation occurs for a remap mismatch. If the request was made to one of the locked blocks, the remap entry is checked. If it matches, then the corresponding subblock is fetched from NM. If not, then the subblock is swapped from FM to NM blocks other than this locked block. Due to space constraint, we do not show the bypassing operation, but for every swap from FM operation in the figure, the access rate is checked. If it is enabled, then the swap from FM becomes a fetch from FM without any metadata update. Also, only the correct way speculation path is shown. In the case of the way prediction misspeculation, the remap entry check takes longer as four remap entries are checked in serial.

IV. Experimental Setup

A. Simulation Infrastructure

To evaluate the SILC-FM scheme, we use a Pin-based simulator that models a 8-core server class processor similar to AMD's A10-7850K processor [35]. Each core is a 4-wide issue processor with 128 ROB entries per core. We also use a detailed memory simulator, Ramulator [36], with 32-entry read and write queues per channel. Timing and configuration parameters are listed in Table II. We implement a virtual-to-physical address translation (2KB page size) and assume that FM to NM capacity ratio is 4:1. For NM memory, we use HBM generation 2 technology and derived timing parameters from JEDEC 235A datasheet [5]. DDR3 technology is used as FM memory with latency parameters derived from JEDEC and vendor datasheet [37], [38]. We also perform the NM capacity sensitivity study in Section V-C. We use the system without NM as baseline scheme. The bit vector history table is 72KB and the predictor is 1.5KB, both of which take 1 cycle. The access latency of SRAM structures is derived from CACTI [39]. We have experimentally found that the threshold of 50 works the best to determine the

Processor	Values
Number of Cores (Freq)	16 (3.2GHz)
Width	4 wide out-of-order
Caches	**Values**
L1 I-Cache (private)	64 KB, 2 way, 4 cycles
L1 D-Cache (private)	16 KB, 4 way, 4 cycles
L2 Cache (shared)	8 MB, 16 way, 11 cycles
HBM	**Values**
Bus Frequency	800 MHz (DDR 1.6 GHz)
Bus Width	128 bits
Channels	8
Ranks	1 Rank per Channel
Banks	8 Banks per Rank
Row Buffer Size	8KB (open-page policy)
tCAS-tRCD-tRP-tRAS	7-7-7-28 (memory cycles)
DDR3	**Values**
Bus Frequency	800 MHz (DDR 1.6 GHz)
Bus Width	64 bits
Channels	4
Ranks	1 Rank per Channel
Banks	8 Banks per Rank
Row Buffer Size	8KB (open-page policy)
tCAS-tRCD-tRP-tRAS	11-11-11-44 (memory cycles)

Table II: Experimental Parameters

block hotness, so we use this value throughout the paper. The execution time is calculated using the time when all workloads in all cores terminate. Our speedup (figure of merit) is calculated using the total execution of the baseline with no HBM memory over the execution time of a corresponding scheme, and therefore, higher speedup represents higher performance.

We compare our scheme against other five other designs: Random Static Placement (rand), HMA (hma), CAMEO (cam), CAMEOP (camp), Part of Memory (pom). Random uses the entire NM and FM as OS visible address space and maps pages randomly. Thus, this scheme does not consider different bandwidth/latency characteristics of NM and FM, and rather, treats them the same. HMA and CAMEO are described in Section II. CAMEOP is not published from prior work, yet we implemented a prefetcher that fetches extra 3 lines along with the miss (we have experimentally found that next 3 lines achieve the best overall performance). Lastly, Part of Memory is implemented based on [28], which migrates 2KB blocks based on block access counts.

B. Workloads

We run a representative region of SPEC CPU2006 benchmark suite using 1 billion instruction slice Simpoint [40]. We choose a subset of benchmarks within the suite, which exhibit high memory bandwidth usage. We run those in a multiprogrammed simulation mode where one copy of the benchmark instance is run on each core. Since each benchmark is a single-threaded application, in our 16-core simulation environment, 16 separate instances of workloads are run with a total of 16 billion instructions across all cores. SPEC calls such simulation rate mode and it is a recommended method to evaluate multicore systems. A benchmark instance on each core does not get migrated to other cores and will stay on its own core until the end of execution. Since we use the multiprogrammed simulation, our virtual-to-physical mapping ensures that different instances do not share the same physical address space.

Category	Benchmark	MPKI (per core)	Footprint (GB)
Low MPKI	bwaves	10.12	6.82
	cactus	7.52	2.31
	dealII	4.46	0.69
	xalanc	5.98	2.87
Medium MPKI	gcc	31.23	1.34
	gems	15.95	10.59
	leslie	11.28	1.19
	omnet	27.22	2.06
	zeusmp	11.41	3.32
High MPKI	lbm	53.29	6.30
	lib	35.50	0.50
	mcf	88.95	18.46
	milc	34.13	9.05
	soplex	43.32	0.78

Table III: Workload Descriptions

With chosen benchmarks, we categorize them into three groups: low, medium, and high Misses Per Kilo Instructions (MPKI). We compute the MPKI based on the number of LLC misses. We place those workloads whose MPKI is lower than 11 as low MPKI benchmarks, those higher than 32 as high MPKI workloads, and those in between as medium MPKI workloads. Table III summarizes our workload composition and related characteristics. All reported MPKI is the LLC MPKI computed per core, and the footprint is calculated by counting the total number of unique pages seen from LLC misses.

V. Evaluation

In this section, unless otherwise specified, all results in the following subsections are normalized with a common baseline of a system without die-stacked DRAM.

A. Performance Analysis

We show the breakdown of SILC-FM execution time improvement in Figure 6. The stack bar begins with the Random scheme as it is the most naive scheme. We show performance improvement achieved through each technique used. SILC-FM swap shows the performance improvement achieved with a direct-mapped small block scheme when any associativity, locking or bypassing technique is not applied. The system is able to achieve a speedup of 1.55 only with the subblock granularity swapping between FM and NM. In high MPKI workloads where more bandwidth demand exists, the swapping alone can significantly alleviate the bandwidth bottleneck by swapping many hot blocks into NM. For that reason, Figure 6 shows that high MPKI workloads achieve the overall highest performance improvement. Workloads such as milc does not get much benefit from swapping as conflicts constantly swap out recently swapped in subblocks, which in turn shows the need for other features that are incorporated in SILC-FM. Now, the SILC-FM adds the locking feature, which can improve the hot page residency in NM. In this case, not all benchmarks benefit as some benchmarks do not have significant number of thrashing or conflicts from the baseline. However, a benchmark such as xalancbmk achieves an extra 14% performance improvement just by locking hot pages. This problem arises due to the fact

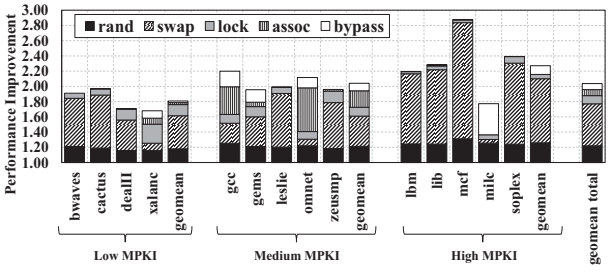

Figure 6: Performance Improvement Breakdown

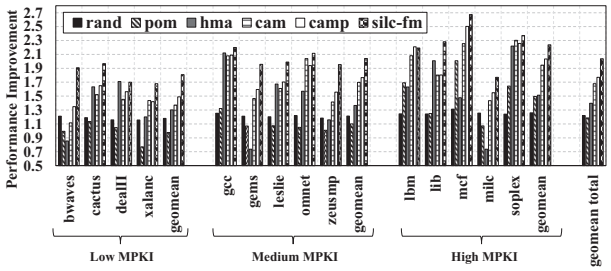

Figure 7: Performance Comparison with Other Schemes

that address bits are used to place blocks, so not all hot pages are evenly spread out in the NM indexing. The xalancbmk benchmark is a good example and locking ensures that some hot blocks are locked, so at least those blocks can be serviced from NM if not all hot blocks can be accommodated in the NM set.

Adding associativity achieves similar effects for unlocked blocks. In some benchmarks, the fraction of hot blocks that reach over the hotness threshold may not be a large portion of the entire working set, meaning many blocks are just lukewarm. In this case, the effects of associativity, which protects those unlocked pages from unwanted conflicts and thrashing, can be quite significant. For example, gcc achieves a significant speedup of 36% with the addition of associativity while locking only improves performance by 11%. This is a good example where the benchmark has many lukewarm blocks. As a result, locking, which only benefits hot blocks, provides negligible improvement, yet associativity brings a huge performance improvement. Lastly, the bypassing feature is added in SILC-FM. Note that the bypassing feature is enabled only when the access rate exceeds 0.8. Benchmarks such as bwaves do not reach this point, so adding the bypassing feature does not provide additional performance. Yet, milc exceeds the 0.8 access rate that the bypassing feature enhances performance by utilizing the FM bandwidth, which otherwise would have been idle. Overall, SILC-FM is able to capture hot blocks and subblocks within lukewarm blocks in NM through features such as swapping, locking, and associativity while higher system-wide bandwidth utilization is achieved through bypassing.

B. Comparison with Other Schemes

Figure 7 shows the performance improvement of our proposed scheme against other schemes. First, the Random scheme does not see much significant performance improvement. The placement is done randomly without considering NM and FM characteristics, so pages are statically allocated. Although some pages may sit in NM, the access rate is low. Since there is no other overheads due to page migration during the execution, all workloads achieve similar performance improvement.

The HMA and PoM scheme improve upon the Random scheme by intelligently selecting hot pages and placing them in NM. The threshold based decision is able to select a subset of pages (mostly hot pages) and move them to NM. As seen in Figure 7, HMA achieves significantly higher performance than Random even though this scheme has additional software overheads such as context switching and TLB shootdowns. This makes the majority of hot pages reside in NM. However, the migration only occurs at a very large time interval, so the selected pages may not be hot anymore by the time the decision is made, which is shown in bwaves and milc. Also, PoM follows the similar trend to HMA, yet in our evaluation, we see that PoM uses much more FM bandwidth as it frequently transfers 2KB blocks. In high spatial workloads, most of 2KB blocks are used, yet in most workloads, we have found that the number of used unique subblocks within 2KB is rather low. Also, PoM has to accumulate a certain access count until the migration is triggered, so it achieves a lower performance.

The CAMEO scheme reacts quickly to any changes in the hot working set and it moves data at a small block granularity. In all workloads, this scheme effectively places most hot small blocks in NM. Yet, the conflict misses are unavoidable since the mapping is direct mapped. For example, cactus suffers from conflict misses, so that schemes such as HMA that can withstand conflicts perform better than CAMEO. In addition, since only one small block is brought into NM at each time, this does not take advantage of abundant spatial locality within a page. However, the CAMEO scheme's data movement granularity of a small block uses FM bandwidth efficiently, so it achieves an overall higher performance improvement than other schemes. The improved CAMEO with prefetcher achieves a higher speedup as it enjoys some degree of spatial locality. However, naively prefetching subblocks also wastes bandwidth as those prefetched subblocks are not always useful.

The SILC-FM scheme effectively removes conflicts by offering associativity and locked blocks. Unlike HMA where pages are migrated and locked into NM at epoch boundaries, the blocks are locked in NM as soon as the access count reaches the threshold. This makes the hot block capturing ability of our scheme respond more quickly to changes in the hot working set. The gemsFDTD workload shows per-

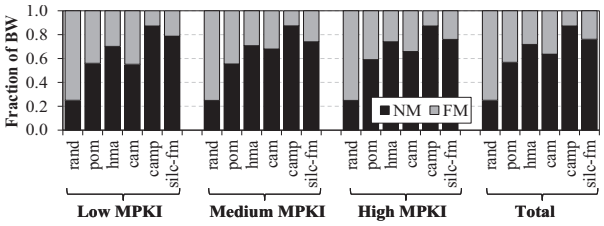

Figure 8: Fraction of FM and NM Bandwidth Usage

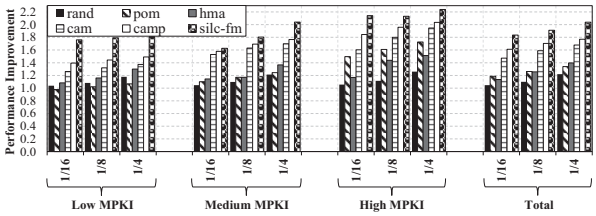

Figure 9: Performance Improvement with Various NM Capacities

formance degradation with HMA, but performance improvement with CAMEO. This benchmark has many short-lived hot pages and the epoch length is too long to make migration decisions. SILC-FM also responds quickly, so performance benefits are seen. The associativity reduces conflicts among those pages that are not locked yet. For libquantum, HMA performs well since it offers fully associative placement at epoch boundaries. Here, CAMEO suffers from conflicts and SILC-FM can withstand that by locking and associativity. Furthermore, our bit vector based fetching scheme migrates more useful subblocks than CAMEO with prefetcher, and thus our benefits are higher. This additional performance gain makes SILC-FM achieve higher performance than the state-of-the-art scheme, CAMEO. Furthermore, the bypassing feature makes certain workloads such as milc extract performance opportunities by using FM bandwidth, which would have been idle in other schemes due to its high access rate.

Figure 8 shows the fraction of the total demand bandwidth usage broken down by either NM or FM. The ideal point here is 0.8 as discussed in Section III-E. Note that here we are only showing the bandwidth consumed by demand requests and not migrations, so the bulk page migration in HMA scheme is not shown here. In HMA and PoM, 71% and 58% of the total demand bandwidth usage on average is consumed by NM, so the NM's high bandwidth is well utilized. Yet, CAMEO's low access rate makes it service more requests from FM when more of NM's bandwidth can be used. CAMEO with prefetcher adds additional traffic to NM bandwidth as prefetched subblocks consume bandwidth, and thus, it creates an imbalance between FM and NM. Without bypassing, SILC-FM leaves FM bandwidth near idle, but by enabling the bypassing feature, we control the access rate. This makes our scheme have 76% of the total bandwidth usage on NM bandwidth, which is only 4% below the ideal of 80%. Therefore, our scheme effectively utilizes available bandwidth in both NM and FM and improves performance.

C. Capacity Analysis

In this section, we analyze the performance improvement with different NM to FM capacity ratios as the proportion of the NM capacity can range from a small to a large fraction of the overall memory capacity. We vary the FM:NM capacity ratio from 1/16 to 1/4. In the recent Intel Knights Landing processor [41], the NM to FM ratio is approximately 1:24, so we sweep the capacity close to this ratio to see performance effects. Figure 9 shows the performance improvement with various schemes and capacities. CAMEO and CAMEO with prefetcher perform better with a larger capacity since it has a larger number of sets, which reduces its inherent problem with conflicts. HMA and PoM's performance is not affected significantly by NM capacities. Although they can capture long lasting hot pages, short living hot pages are not well captured. The set of benchmarks used in this paper does not have the varying number of long lasting pages with NM capacities, and thus, their performance remains approximately constant with varying NM capacities. However, in other application domains such as cloud computing, it is possible that their performance improvement can be noticeable. SILC-FM, on the other hand, captures hot blocks/subblocks across various capacities. When the capacity is small, such as 1/16, the scheme also has a smaller number of sets as in the case of CAMEO, yet the locking and associativity significantly improve such problems. This is apparent in low MPKI workloads, which does not see significant performance degradation with much reduced number of sets. Therefore, when conflicts and thrashing play a huge performance role in other schemes such as CAMEO, SILC-FM is able to minimize such impact when the capacity is reduced. Overall, SILC-FM is able to provide an average performance improvement from 1.83 to 2.04 when the ratio grows from 1/16 to 1/4 while the best comparison scheme can provide only from 1.47 to 1.67.

VI. Related Work

Emerging memory technologies have provided opportunities for creating interesting memory system designs. Much of the work has been focused on efficiently storing metadata overheads for scalability of this multi-gigabyte memory technology. Since memory bandwidth is often the bottleneck, some work[11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22] has focused on reducing the FM bandwidth usage while managing high hit rate to NM. These proposals manage NM as hardware caches, and thus, do not take advantage of added NM capacity. In near future, the NM capacity is expected to take a considerable amount of the main memory, and thus, some work [7], [28] has focused

on using this capacity as apart of OS visible space to take advantage of added capacity. However, their schemes are based on conventional caching techniques that are susceptible to problems that exist in SRAM caches. Purely software techniques [23], [24], [25], [26], [27], [42], [43] have been proposed, yet the high overheads associated with software intervention achieves suboptimal performance.

Cache bypassing has been done in on-chip caches, but with a goal of reducing conflict misses. Our scheme bypassing is performed to better utilize the overall system bandwidth. Recent work [28], [33] provides a bypassing feature, however, a rather significant amount of NM region is disabled and bypassed. Our bypassing is not tied to any particular region and is more fine-grained (at page granularity). Column caching [44] is one SRAM cache technique that prevents one entire way from being involved in hardware eviction activities, so in the context of 1GB NM, it means to lock 256MB. Also, it is proposed to be managed as a scratch pad where users have to explicitly allocate data, whereas locking in our scheme is transparent to the user.

VII. Conclusion

Prior die-stacked DRAM approaches focused on using block or epoch based schemes, but adopting either one will benefit only a subset of different workloads. In this paper, we have presented an associative locking memory architecture called SILC-FM that locks hot pages in NM and intelligently remaps FM subblocks into NM. Unlike using NM as a cache, SILC-FM fully exposes the die-stacked DRAM capacity to the OS to take advantage of the additional capacity provided by NM. SILC-FM incorporates a predictor to reduce the access latency. In addition, some memory requests bypass NM and directly access FM to utilize the FM bandwidth available to the processor. In the end, SILC-FM is able to achieve on average 36% performance improvement over state-of-the-art die-stacked DRAM architecture.

References

[1] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki, "Toward dark silicon in servers," *IEEE Micro*, vol. 31, no. EPFL-ARTICLE-168285, pp. 6–15, 2011.

[2] B. M. Rogers, A. Krishna, G. B. Bell, K. Vu, X. Jiang, and Y. Solihin, "Scaling the bandwidth wall: challenges in and avenues for cmp scaling," in *ACM SIGARCH Computer Architecture News*, vol. 37, no. 3. ACM, 2009, pp. 371–382.

[3] HMC Consortium, "HMC Specification 1.0," http://www.hybridmemorycube.org, 2012.

[4] JEDEC, "High Bandwidth Memory (HBM) DRAM (JESD235)," https://www.jedec.org, 2013.

[5] ——, "High Bandwidth Memory (HBM) DRAM Gen 2 (JESD235A)," https://www.jedec.org, 2016.

[6] J. T. Pawlowski, "Hybrid Memory Cube (HMC)," in *Proceedings of 23rd Hot Chips*, 2011.

[7] C. Chou, A. Jaleel, and M. K. Qureshi, "CAMEO: A Two-Level Memory Organization with Capacity of Main Memory and Flexibility of Hardware-Managed Cache," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-47, 2014.

[8] G. B. Berriman, G. Juve, E. Deelman, M. Regelson, and P. Plavchan, "The application of cloud computing to astronomy: A study of cost and performance," in *e-Science Workshops, 2010 Sixth IEEE International Conference on*. IEEE, 2010, pp. 1–7.

[9] Y. Chen, S. Alspaugh, and R. Katz, "Interactive analytical processing in big data systems: A cross-industry study of mapreduce workloads," *Proc. VLDB Endow.*, vol. 5, no. 12, pp. 1802–1813, Aug. 2012. [Online]. Available: http://dx.doi.org/10.14778/2367502.2367519

[10] C. Luo, J. Zhan, Z. Jia, L. Wang, G. Lu, L. Zhang, C.-Z. Xu, and N. Sun, "Cloudrank-d: benchmarking and ranking cloud computing systems for data processing applications," *Frontiers of Computer Science*, vol. 6, no. 4, pp. 347–362, 2012.

[11] M. El-Nacouzi, I. Atta, M. Papadopoulou, J. Zebchuk, N. E. Jerger, and A. Moshovos, "A dual grain hit-miss detector for large Die-Stacked DRAM caches," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, 2013, pp. 89–92.

[12] S. Franey and M. Lipasti, "Tag tables," in *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, 2015.

[13] N. Gulur, M. Mehendale, R. Manikantan, and R. Govindarajan, "Bi-Modal DRAM Cache: Improving Hit Rate, Hit Latency and Bandwidth," in *Microarchitecture (MICRO), 2014 47th Annual IEEE/ACM International Symposium on*, 2014.

[14] F. Hameed, L. Bauer, and J. Henkel, "Simultaneously Optimizing DRAM Cache Hit Latency and Miss Rate via Novel Set Mapping Policies," in *Proceedings of the 2013 International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, ser. CASES '13, 2013.

[15] D. Jevdjic, G. Loh, C. Kaynak, and B. Falsafi, "Unison Cache: A Scalable and Effective Die-Stacked DRAM Cache," in *Microarchitecture (MICRO), 2014 47th Annual IEEE/ACM International Symposium on*, 2014.

[16] D. Jevdjic, S. Volos, and B. Falsafi, "Die-stacked DRAM Caches for Servers: Hit Ratio, Latency, or Bandwidth? Have It All with Footprint Cache," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ser. ISCA '13, 2013.

[17] X. Jiang, N. Madan, L. Zhao, M. Upton, R. Iyer, S. Makineni, D. Newell, Y. Solihin, and R. Balasubramonian, "CHOP: Integrating DRAM Caches for CMP Server Platforms," *IEEE Micro*, vol. 31, no. 1, 2011.

[18] G. Loh and M. D. Hill, "Supporting Very Large DRAM Caches with Compound-Access Scheduling and MissMap," *IEEE Micro*, vol. 32, no. 3, May 2012.

[19] J. Meza, J. Chang, H. Yoon, O. Mutlu, and P. Ranganathan, "Enabling Efficient and Scalable Hybrid Memories Using Fine-Granularity DRAM Cache Management," *Computer Architecture Letters*, vol. 11, no. 2, pp. 61–64, July 2012.

[20] 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*, ser. MICRO-45, 2012.

[21] J. Sim, G. H. Loh, H. Kim, M. O'Connor, and M. Thottethodi, "A Mostly-Clean DRAM Cache for Effective Hit Speculation and Self-Balancing Dispatch," in *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-45, 2012.

[22] L. Zhao, R. Iyer, R. Illikkal, and D. Newell, "Exploring DRAM cache architectures for CMP server platforms," in *Computer Design, 2007. ICCD 2007. 25th International Conference on*, 2007, pp. 55–62.

[23] X. Dong, Y. Xie, N. Muralimanohar, and N. P. Jouppi, "Simple but Effective Heterogeneous Main Memory with On-Chip Memory Controller Support," in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '10, 2010.

[24] G. H. Loh, N. Jayasena, K. Mcgrath, M. O'Connor, S. Reinhardt, and J. Chung, "Challenges in Heterogeneous Die-Stacked and Off-Chip Memory Systems," in *the 3rd Workshop on SoCs, Heterogeneous Architectures and Workloads (SHAW)*, 2012.

[25] M. Meswani, S. Blagodurov, D. Roberts, J. Slice, M. Ignatowski, and G. Loh, "Heterogeneous memory architectures: A HW/SW approach for mixing die-stacked and off-package memories," in *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, 2015.

[26] L. E. Ramos, E. Gorbatov, and R. Bianchini, "Page Placement in Hybrid Memory Systems," in *Proceedings of the International Conference on Supercomputing*, ser. ICS '11, 2011.

[27] Z. Wang, D. Jimenez, C. Xu, G. Sun, and Y. Xie, "Adaptive placement and migration policy for an stt-ram-based hybrid cache," in *High Performance Computer Architecture (HPCA), 2014 IEEE 20th International Symposium on*, 2014.

[28] J. Sim, A. R. Alameldeen, Z. Chishti, C. Wilkerson, and H. Kim, "Transparent Hardware Management of Stacked DRAM as Part of Memory," in *Proceedings of the 2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-47, 2014.

[29] S. M. Khan, D. A. Jiménez, D. Burger, and B. Falsafi, "Using dead blocks as a virtual victim cache," in *Proceedings of the 19th international conference on Parallel architectures and compilation techniques*. ACM, 2010, pp. 489–500.

[30] S. Somogyi, T. F. Wenisch, A. Ailamaki, and B. Falsafi, "Spatio-temporal memory streaming," in *ACM SIGARCH Computer Architecture News*, vol. 37, no. 3. ACM, 2009, pp. 69–80.

[31] G. Tyson, M. Farrens, J. Matthews, and A. R. Pleszkun, "A modified approach to data cache management," in *Proceedings of the 28th annual international symposium on Microarchitecture*. IEEE Computer Society Press, 1995, pp. 93–103.

[32] C.-J. Wu, A. Jaleel, W. Hasenplaugh, M. Martonosi, S. C. Steely Jr, and J. Emer, "Ship: Signature-based hit predictor for high performance caching," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2011, pp. 430–441.

[33] C. Chou, A. Jaleel, and M. K. Qureshi, "Bear: techniques for mitigating bandwidth bloat in gigascale dram caches," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture*. ACM, 2015, pp. 198–210.

[34] N. Agarwal, D. Nellans, M. O'Connor, S. W. Keckler, and T. F. Wenisch, "Unlocking bandwidth for gpus in cc-numa systems," in *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*. IEEE, 2015, pp. 354–365.

[35] AMD, "AMD A-Series Desktop APUs," http://www.amd.com/en-us/products/processors/desktop/a-series-apu.

[36] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A Fast and Extensible DRAM Simulator," *Computer Architecture Letters*, vol. PP, no. 99, pp. 1–1, 2015.

[37] JEDEC, "DDR SDRAM (JESD79-3C)," https://www.jedec.org, 2008.

[38] Micron Technology Inc., "TN-46-03 Calculating Memory System Power for DDR," 2001.

[39] P. Shivakumar and N. P. Jouppi, "Cacti 3.0: An integrated cache timing, power, and area model," Technical Report 2001/2, Compaq Computer Corporation, Tech. Rep., 2001.

[40] E. Perelman, G. Hamerly, M. Van Biesbrouck, T. Sherwood, and B. Calder, "Using SimPoint for Accurate and Efficient Simulation," *SIGMETRICS Perform. Eval. Rev.*, vol. 31, no. 1, Jun. 2003.

[41] Intel, "KnightsLanding," http://www.realworldtech.com/knights-landing-details.

[42] 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, no. 3. ACM, 2015, pp. 211–222.

[43] H. Jang, Y. Lee, J. Kim, Y. Kim, J. Kim, J. Jeong, and J. W. Lee, "Efficient footprint caching for tagless dram caches," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2016, pp. 237–248.

[44] D. Chiou, P. Jain, L. Rudolph, and S. Devadas, "Application-specific memory management for embedded systems using software-controlled caches," in *Proceedings of the 37th Annual Design Automation Conference*. ACM, 2000, pp. 416–419.