MemPod: A Clustered Architecture for Efficient and Scalable Migration in Flat Address Space Multi-level Memories

Andreas Prodromou[1], Mitesh Meswani[2*], Nuwan Jayasena[3], Gabriel Loh[3] and Dean M. Tullsen[1]
[1]University of California, San Diego
[2]ARM
[3]AMD Research

ABSTRACT

In the near future, die-stacked DRAM will be increasingly present in conjunction with off-chip memories in hybrid memory systems. Research on this subject revolves around using the stacked memory as a cache or as part of a flat address space. This paper proposes MemPod, a scalable and efficient memory management mechanism for flat address space hybrid memories. MemPod monitors memory activity and periodically migrates the most frequently accessed memory pages to the faster on-chip memory. MemPod's partitioned architectural organization allows for efficient scaling with memory system capabilities. Further, a big data analytics algorithm is adapted to develop an efficient, low-cost activity tracking technique.

MemPod improves the average main memory access time of multi-programmed workloads, by up to 29% (9% on average) compared to the state of the art, and that will increase as the differential between memory speeds widens. MemPod's novel activity tracking approach leads to significant cost reduction (∼12800x lower storage space requirements) and improved future prediction accuracy over prior work which maintains a separate counter per page.

Keywords: Memory architecture, Die-stacked memory

1. INTRODUCTION

Die-stacked memory will increasingly be part of future systems, attempting to alleviate the memory wall [1]. Memory standards have been developed [2, 3, 4] and processor manufacturers are already announcing products featuring 3D-stacked memory [5, 6, 7, 8]. Currently, this technology is limited to 8GB per stack [9] which does not fully address the capacity demands of modern systems. Therefore, die-stacked memories are expected to coexist with larger, slower off-chip memories, such as DDR4 or emerging byte addressable NVRAM [10, 11, 12] technologies, in a configuration often referred to as "Two-Level Memory" (TLM) [13, 14].

Stacked memory can be used as a large, high-bandwidth last level cache, or as part of main memory in a "flat address space". When used as a cache, recent research [15, 16] demonstrates the need to re-evaluate traditional SRAM-based cache organizations. Tag placement and granularity need to be re-evaluated, and we should avoid double memory accesses for tag check and data retrieval. Managing stacked memory as a cache is transparent to the software and improves the performance of latency-sensitive applications. However, capacity-sensitive applications do not gain significant improvements as the stacked memory's (typically significant) capacity is not utilized for additional storage.

In a flat address space configuration, the capacity of stacked memory can be allocated and used by applications. Dynamic memory managers proposed in the literature [17, 14, 13] monitor and profile memory accesses and attempt to transparently migrate frequently accessed pages to the fast portion of memory. While exposing more memory to the system, profiling memory accesses and performing transparent migrations often come with power, performance, and space overheads.

Software migration schemes [14] have high performance overheads and operate at coarse intervals, and thus are slow to adapt to changes in application phases. Recently proposed hardware managed schemes [17, 13] operate at finer granularity than software by either using simple, cache-like demand-driven migration or using a centralized management scheme. The former does not consider "hotness" of data, whereas the latter will not scale to large memories due to its centralized approach.

This paper introduces MemPod, a dynamic memory manager for flat address space memory configurations that is area efficient and high performance. It scales particularly well to future technologies with higher memory technology performance differentials. MemPod's novel microarchitectural design clusters existing memory controllers into memory "Pods". Each Pod operates independently and in parallel allowing for better scalability and integration to future systems with larger and faster memories with possibly a higher number of channels. For MemPod's activity tracking requirements we incorporate the *"Majority Element Algorithm" (MEA)* heuristic [18, 19], originally proposed for database management and big data analytics. Our evaluation shows MEA to be capable of high prediction accuracy with very low hardware overhead. To the best of our knowledge, MEA has not previously been proposed for activity tracking in hardware.

*Worked on this project while employed at AMD Research.

2378-203X/17 $31.00 © 2017 IEEE
DOI 10.1109/HPCA.2017.39

Our evaluation results with homogeneous and mixed 8-core multi-programmed workloads show MemPod to outperform the current state-of-the-art by 9% on average and up to 29% in terms of Average Main Memory Access Time (AMMAT) (i.e. the average time a request spends waiting for main memory). Modeling future memory configurations, our results show MemPod to be the most scalable mechanism as memory technology improves. The use of MEA activity tracking requires $\sim 0.01\%$ of the storage space required by the Full Counters (FC) approach used in previous research studies which uses one access counter per memory page or region, while at the same time achieving more accurate prediction of future hot pages.

We identify the fundamental building blocks of any flat address space dynamic memory management mechanism, and describe the solution for each block in prior proposed systems and MemPod, along with their various tradeoffs.

The contributions of this paper are:

- Novel activity tracking algorithm (Section 3).
- Breakdown of the basic building blocks of a flat address space dynamic memory management mechanism (Section 4).
- Novel clustered microarchitecture (Section 5).
- Evaluation of MemPod's effectiveness and its sensitivity to design parameters (Section 6).

2. RELATED WORK

A wide range of research proposals have sought to address the memory wall. Techniques such as *Bump[20], RMM[21] and Superpages[22]* attempt to optimize page placement in memory to expose higher parallelism. However, these scheduling mechanisms do not take advantage of a faster memory in a hybrid configuration.

Stacking DRAM dies in the processor package has been shown to achieve significant performance improvement. This technology cannot yet deliver large capacities [9]. Consequently, configurations combining stacked and off-chip memories have been proposed [23, 16] and can be found in the literature as "hybrid memories" or "two-level memories". The systems have proposed the use of the stacked memory either as a large high-bandwidth last level cache or as a "flat address space", where the capacity of the stacked memory is exposed to the software.

Organizing stacked memory as a cache has been explored in several studies [16, 24, 25, 26, 27, 28]. These approaches implement intelligent tag stores to allow cache-like operation while mitigating the cost of reading tags in DRAM. It has been demonstrated that traditional SRAM-tailored cache optimizations result in degraded performance when used in a DRAM cache and as such we need to "de-optimize for performance" [16]. DRAM cache organizations have been shown to improve performance significantly in latency-limited applications, while offering only marginal improvement with capacity-limited applications. It's been shown that exposing the extra capacity to the application instead of using it as a cache can benefit capacity-limited applications. To this end, recent work [14, 17, 13] proposes mechanisms to manage stacked memory as a flat address space.

HMA [14] is a HW/SW mechanism that attempts to predict frequently accessed pages in memory and, at predefined intervals, migrate those pages to fast memory. HW support is required for profiling memory accesses using counters for each memory page, while the migration is handled by the OS. Due to the costly OS involvement, HMA's intervals are kept large. Additionally, the hardware cost of its profiling counters is high. However, HMA is capable of managing migrations in a flat address space without the need of additional bookkeeping for finding migrated pages as the OS can update page tables and TLBs to reflect migrations.

Sim, et al. proposed a technique for transparent hardware management of a hybrid memory system [17], which we will refer to as "THM". THM does not require OS intervention while managing migrations. In order to keep bookkeeping costs manageable, THM allows migrations only within sets of pages (called segments). Each segment includes one fast memory page and a set of slow memory pages. The slow pages of each segment can only migrate to the one fast page location, and any such migration results in the eviction of the currently-residing page. THM monitors memory accesses with one "competing counter" per segment resulting in a low cost profiling solution. Finally, THM supports caching part of its structures on chip while the rest is stored in memory.

CAMEO [13] proposes a cache-like flat address space memory management scheme in an attempt to close the gap between cache and flat memory organizations. CAMEO operates similarly to THM, however it does so at the granularity of cache lines (64B). Migrations are restricted within segments with one fast line location per segment. Its bookkeeping structures are entirely stored in memory, while a "Line Location Predictor" attempts to save some bookkeeping-related accesses by predicting the location of a line. CAMEO initiates a line migration upon every access to slow memory.

Both THM and CAMEO sacrifice migration flexibility for area efficiency by restricting migrations in segments: if more than one hot page/line exists within the same segment only one can reside in fast memory. If no hot pages exist in a segment, its fast page cannot be utilized by another segment. Further, THM's competing counters can lead to false positives, allowing a cold page to migrate to fast memory, while CAMEO can incur high migration traffic as every access could induce a migration.

Spatial locality of applications can affect performance negatively when THM or CAMEO are used. Continuous pages or lines that lie within the same segment of each mechanism can be accessed frequently. THM is less susceptible to such issues because of its coarser granularity and the use of competing counters that will prevent a "ping-pong" effect. CAMEO, however is significantly affected. This issue is further exacerbated

Algorithm 1: Majority Element Algorithm

Input: X: Set of N elements
Input: K: Number of elements to output
Data: T: Map structure with K entries
Result: Set of K majority elements

Initialization: $T \leftarrow \emptyset$

foreach $i \in X$ **do**
  if $i \in T$ **then**
    $T[i] \leftarrow T[i] + 1$;
  else if $|T| < K - 1$ **then**
    $T[i] = 1$;
  else
    forall $j \in T$ **do**
      $T[j] \leftarrow T[j] - 1$;
      if $T[j] == 0$ **then** $T \leftarrow T \setminus j$;
    end
  end
end

when the ratio between slow and fast memory capacities is increased. In such scenarios, under a configuration with 1:8 fast:slow memory ratio, both mechanisms suffer from reduced migration flexibility, e.g. forcing 8 slow pages to fight over a single fast page or line. In CAMEO's case, since every access to a slow line triggers a migration, a high slow-fast capacity ratio can result in the majority of accesses going to slow lines, causing a migration in every case.

3. PREDICTING HOT REGIONS WITH MEA

Migration mechanisms predict *future* hot pages to migrate them into fast memory. Prediction accuracy is critical to high performance, as each migration must be amortized by many future accesses to justify the cost of migration. A commonly used practice is to identify the hot regions within an interval and assume that those regions will be hot in the next interval. To accurately identify the hottest regions some mechanisms use an access counter per region. At the end of each interval the counters are sorted to identify the highest ranked (i.e., most accessed) regions. However, application phase changes could render this approach unsuccessful. Additionally, the number of necessary counters increases linearly as memory capacities grow.

To address the above limitations, we adopt a technique based on the Majority Element Algorithm (MEA). MEA was originally proposed by Karp et al. [18] and was studied in-depth by Charikar et al. [19] for database management and big data analytics. This heuristic has formally been proven to correctly identify the K most frequently occuring elements of a set, when each of those elements appears more than $\frac{N}{K+1}$ (i.e. has majority), where N is the number of elements in the input set.

MEA is presented in Algorithm 1 as applied to an array of integers X. A map structure T maps K element IDs (in our integer array example, IDs are the integers' values) to K counters. Looping through the array, if the next integer exists in the map, its counter is incremented by 1. Otherwise, if there's enough room in the map a new entry is added with a count of 1. If the number does not exist in the map and all K counters are occupied, MEA subtracts 1 from every counter, removes the entries with a counter value of 0 and proceeds to the next integer. Once the entire array is processed, the map entries hold the majority elements.

A hardware implementation of MEA requires a mapping from page IDs to counters, as well as the area overhead of the counters themselves. On each access, the algorithm will perform one of the following operations: (a) find and increase exactly one counter by one, (b) add a new entry and set its counter value to 1 or (c) subtract one from all counters regardless of the page ID field and identify all the entries with a counter value of zero. All possible operations are simple and can complete within a single cycle if designed properly, using parallel subtractions and comparisons for operation (c).

In our application of MEA to activity tracking, the sequence of page addresses accessed correspond to the array of integers in the above example. However, we find that the sequence of accessed pages typically does not meet the condition of MEA that guarantees it will find the most-accessed pages; thus, it becomes an approximation. What makes MEA most useful, though, is its failure mode – when it fails to find the most-accessed pages, it does so by favoring recency over quantity. That is, a page accessed several times near the end of an interval can easily knock out a page accessed many more times early in the interval. As a result, it combines both access counting and temporal locality, at a fraction of the cost of access counting alone.

MEA's area overhead grows slowly with the amount of memory per pod. The number of counters can be kept constant, but the size of the ID or tag grows with the log of the memory size. Its $O(N)$ complexity works well for analyzing a stream of access requests in real time, and eliminates the need for sorting the counters.

In this section, we seek to understand the effectiveness of MEA's counting and prediction accuracy, compared against a Full Counters (FC) scheme, independent of the MemPod architecture. We use memory traces captured from multi-programmed 8-core workloads (the same traces used and described in Section 6) and simulate MEA and FC side-by-side with an in-house off-line simulator that provides oracle knowledge of future intervals.

The interval size for both MEA and FC was set at 5500 requests which is the average number of requests serviced within a 50us window in our timing experiments. For this experiment we use 128 MEA counters and FC requires 4.5M counters (assuming a 1+8GB memory capacity). We do two comparisons in this study. First, we examine the ability of MEA to identify the top pages in the past interval, something the full counter

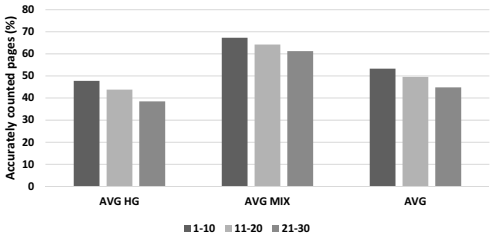


Figure 1: MEA counting accuracy compared to Full Counters on the top three tiers (ranks 1-10, 11-20, 21-30). Average results for homogeneous(AVG HG), mixed (AVG MIX) and all (AVG ALL) workloads shown.

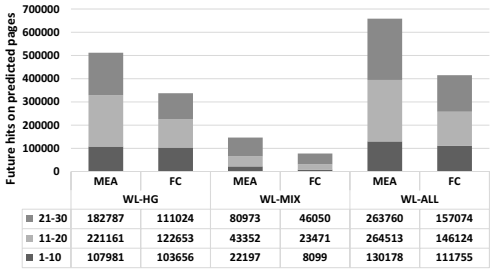


Figure 2: MEA prediction accuracy (part 1) compared to Full Counters on the top three tiers (ranks 1-10, 11-20, 21-30). Results for homogeneous(WL-HG), mixed (WL-MIX) and all (WL-ALL) workloads shown.

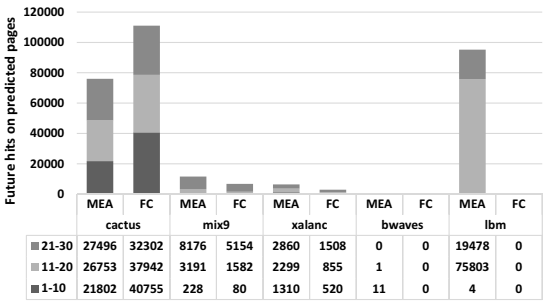


Figure 3: MEA prediction accuracy (part 2). This graph presents the most interesting results from individual workloads.

scheme will do perfectly. Second, we examine the ability of both schemes to predict the top pages in the next interval. In both studies, we will examine the schemes' ability to identify the 30 most-accessed pages, in bins of 10 each (1-10, 11-20, and 21-30).

Figure 1 shows the counting accuracy of MEA for the past interval, which should be compared with FC's perfect accuracy. In some workloads, MEA identified up to 75% of the top pages. However, on average MEA reports accuracy below 55% on the top tiers. Thus, it is a surprisingly ineffective replacement for accurate counters, if accurate counting were our priority. The bias toward recent accesses has a strong effect on the final value of the MEA counters.

When we instead examine the effectiveness in identifying future hot pages, we see a different story. Figure 2 presents a comparison of MEA and FC in terms of prediction accuracy. We compare each mechanism's "predictions" against the top three page tiers of the following interval based on oracular knowledge. Using 128 counters, MEA will return *up to* 128 predictions based on the past interval, while FC will return an overall ranking of each page accessed. In order to be able to directly compare accuracy, we take the top N pages from the full counters each interval, where N is the number of pages MEA returned.

Figure 2 plots the number of hits on predicted hot pages from the previous interval. We also select interesting individual benchmarks and show them in Figure 3 to provide a more detailed comparison.

On average, MEA achieves more future hits than FC by 16%, 81% and 68% on the top three tiers respectively. Figure 3 shows selected individual workloads that generated interesting results and provides a more detailed comparison. Cactus[1] is the only workload where FC outperformed MEA's prediction. In fact it outperformed MEA on every tier.

Xalanc and mix9 are most representative of our overall results. We can see MEA outperforming FC's prediction accuracy in every bin. The last two workloads we selected, bwaves and lbm, show FC failing entirely to predict the future (FC also scored zero future hits with our libquantum workload). With bwaves (and libquantum) MEA reports a very low number of future hits but not zero. These results can happen when an application streams through large structures that exceed the size of the interval. In that case, the past interval has little overlap with the next interval, but recent accesses are much more likely to be overlapped. Lbm shows an interesting result, where MEA reports a high number of hits (outside of the first tier) in a workload where FC failed entirely. This can happen with a large working set where the application does a fairly constant amount of work per page. Full counters, then, will record the highest access counts for pages the application is done with, while MEA will favor pages the application was still working on at the end of the interval.

MEA has not previously been used in any kind of architectural event tracking; however, our results indicate it is an attractive alternative to full counters at a very small fraction of the hardware cost.

[1]We use a single benchmark's name as a shorthand for workloads running the same benchmark 8 times simultaneously on 8 cores.

4. MIGRATION BUILDING BLOCKS

The design of a complete memory manager can be broken down into the following 5 "building blocks":

Migration flexibility: This is defined by the possible mappings in fast memory that a particular memory region (page) can map to.

Remap table: A structure that keeps track of migrated pages and is able to provide a relay address given a requested address.

Activity tracking: Logic and structures needed to pro-

file memory requests and predict future "hot" pages.

Migration trigger: Defines when migration occurs. Commonly the trigger can be event, interval, or threshold-based.

Migration driver/datapath: Defines the path followed and the hardware modules involved in performing migrations.

Each of the above building blocks introduces some trade-off. For example, allowing more flexibility in migration locations can lead to higher performance benefits, at the cost of larger book-keeping structures. The design choices for the various building blocks are largely orthogonal. Thus, architects can select an approach to each building block suitable to their system's capabilities and limitations and simply combine them to create a desirable memory management mechanism.

4.1 Migration Flexibility and Remap Table Size

Migrating pages can provide the highest benefit when no restrictions are imposed on the available migration locations. This amount of flexibility, however, requires more bookkeeping and can incur a higher cost.

A hardware-driven migration mechanism requires some kind of remap table, commonly implemented as a hash structure, indexed by a page's address and pointing to the migrated (or relay) page address if one exists. On a page migration, the remap table is updated to reflect the new address of a migrated page.

The remap table should provide the remap (relay) address for each original page address. Some other structures may also be necessary to avoid expensive table searches when an inverse lookup is needed (e.g., to identify pages currently mapped to slow or fast memory).

4.2 Activity Tracking

Activity tracking is a critical element of any management mechanism for hybrid memories. In most studies on the subject, activity tracking becomes a synonym for identifying hot regions by counting the number of accesses to each one. In a more generalized approach, it could potentially be extended to track patterns, parallelism, bit flips/faults or any other information useful to the underlying mechanism.

The overhead of maintaining a set of counters per memory page (or other granularity) will be high. Space requirements will increase linearly as memory capacities grow and the cost of sorting all the counters can overshadow any potential benefits in performance. Furthermore, our evaluation presented in Section 3 demonstrates that using full counters to ensure 100% accurate counting may still lead to poor prediction accuracy. Frequently encountered cost-reducing solutions in the literature consist of increasing the activity tracking granularity in order to reduce the number of counters needed (i.e. track a group of pages together), limiting the bit width of counters, and caching a subset of the tracking state while the full set resides in main memory.

4.3 Migration Triggers

Each memory management policy must decide when to trigger migrations. Migrations add significant delays to a system and any penalties incurred should be amortized by the performance improvement from placing a page in the fast memory. Requests that arrive while migrations are being performed have to be delayed to ensure functionally correct memory behavior. Throughout the literature, three triggers are most commonly used whenever state must be updated based on tracking information (MC scheduling, migrations, dynamic voltage and frequency scaling etc.). Interval-based (or epoch-based) triggers occur with a set frequency, while threshold-based solutions trigger whenever a predetermined criterion is met. Finally, event-based triggers react to predefined events. Both interval-based and threshold-based approaches face the same challenge of identifying the optimal interval or threshold value.

4.4 Migration Datapath

Regardless of the choice for each migration building block described so far, once migration is triggered, the migration manager has to follow a number of steps: First, migration candidates need to be identified. Traditionally, one page (or a segment/line depending on the migration granularity) from the slow memory and one from fast memory. Then, the two identified candidates need to be swapped. During the swap process, one or both pages will be read and stored in temporary buffers and then written at their remapped locations.

Without dedicated migration driver hardware, migrations will have to be orchestrated by the OS and CPU cores. Consequences include communication delay, potentially some pollution of the processors' caches and the unavailability of those CPUs during migration.

MemPod implements the migration driver within each Pod. As each Pod has direct communication with its member MCs, added delays are kept to a minimum, and no traffic is generated at the global switch (saving energy and eliminating contention). In HMA, the OS orchestrates everything. Some CPU cores have to be stalled and used to service the OS interrupt, causing the migrated pages to traverse through communication mediums and caches on each way. THM does not fully describe its datapath, but it appears that CPUs are used in this case as well. CAMEO describes its swapping operation to be transparent to the OS by using existing writeback and fill queues. Its mechanism relies on the two memories sending writeback or demand read requests to each other, which further implies some added logic in Memory Controllers, as well as communication between MCs.

To make a generous comparison, we do not model the penalty introduced by using CPUs or global communication mediums for migration in our HMA, THM, and CAMEO simulations presented in Section 6.

5. ARCHITECTURE

Our clustered migration mechanism is designed to address key challenges associated with the migration problem. In this section, we first present a high-level

overview of MemPod's proposed micro-architectural design, followed by a more detailed discussion regarding each building block. Throughout this section we also discuss some of the major design decisions of the state-of-the-art mechanisms.

5.1 Clustered Migration Architecture

Figure 4 presents an overview of MemPod. MemPod's design was kept modular to facilitate system integration and scalability. A number of memory "Pods" are injected between the Last Level Cache (LLC) and the system's memory controllers (MCs). Each Pod clusters a number of MCs and enforces migrations to only occur among its member MCs. Pods do not communicate with each other, preventing inter-Pod migrations. To the rest of the system, Pods are exposed as MCs. With MemPod's transparent design, each Pod will now be receiving all the requests originally addressed to any of the Pod's member MCs.

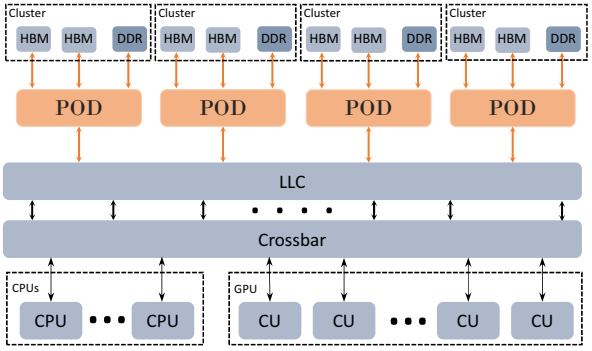


Figure 4: MemPod high-level architecture

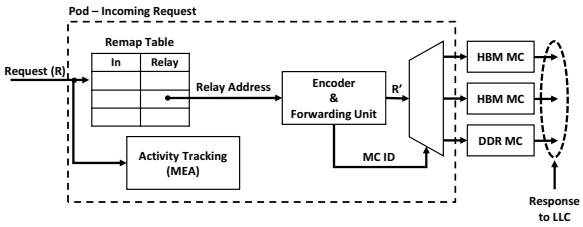
When a memory request arrives, the Pod monitors the request, updates any necessary migration-related activity tracking counters, and forwards the request to the intended recipient MC. The migration logic within a Pod does not need to be invoked during a response from any MC and can be bypassed to reduce memory access latency. A drawback of clustering MCs into Pods is the serialization of potentially parallel requests to different MCs of a single Pod. As such, activity tracking within a Pod as well as the subsequent forwarding of requests must be as efficient as possible.

MemPod's clustered architecture also reduces global traffic during migrations compared to non-partitioned mechanisms. Because migration traffic happens within a Pod, this architecture significantly reduces global traffic and enables parallel migrations.

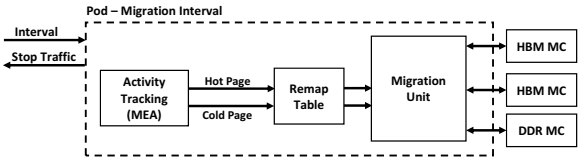
Memory Pod

The major architectural elements of a Pod are shown in Figure 5. A Pod includes an activity tracking (MEA) unit, a remap table for keeping track of migrated pages and a forwarding unit that can re-encode a request with the relay address and, based on that address, send the request to the appropriate MC.

A designer can vary the parallelism and flexibility



(a) Pod's request forwarding operation



(b) Pod's migration procedure

Figure 5: Major architectural Pod elements

of MemPod by varying the number of Pods. A design with one Pod is equivalent to a centralized migration controller allowing any-to-any migration, while a design with a Pod number equal to the number of MCs would imply that migration is disabled. A reasonable design point would be to set the number of Pods equal to the number of slow-memory MCs. Such a configuration inherently prevents migration between slow off-chip channels, while at the same time maintaining full channel-level parallelism on the system's bottleneck: the slow MCs. In a configuration where the number of fast-memory MCs are not a multiple of slow-memory MCs, Pods can be configured asymmetrically or some MCs could be members of multiple Pods, with their capacity partitioned to avoid crosstalk issues. In Figure 4 we present a system with eight MCs for the fast, on-die stacked memory and four MCs for the slow off-chip memory. Throughout this paper we use die-stacked HBM as the fast memory [9] and DDR4-1600 as our off-chip memory, and we set the number of Pods to four, as shown in Figure 4.

5.2 Building Blocks

MemPod imposes few migration restrictions since each slow page can migrate to any fast page location as long as it's within the same Pod. To support high flexibility MemPod requires a Remap Table structure capable of tracking all pages at each Pod and upon a lookup return the new address. The page table is updated to reflect changes with each migration. In addition to the remap table, our algorithm also needs to identify all pages currently mapped to fast memory (to identify a candidate to be evicted in favor of a new hot page). We do this with a smaller, inverted table that gives the original address of each page currently mapped to fast memory.

MemPod requires an MEA map structure of K entries, where K is the number of hot pages we wish to identify at each interval. Our evaluation presented in Section 6 finds a good number of MEA counters to be 64. Each entry maps a page's address to a counter.

Challenge	Tradeoff	THM	HMA	CAMEO	MemPod
Page Relocation	*Flexibility / Time*	Only 1 Candidate	No Restrictions	Only 1 Candidate	Intra-Pod Migration
Remap Table Size	*Flexibility / Area*	1 entry per fast page (1.5kB)	No remap table	1 entry per fast line (72kB)	1 entry per page (2.8 MB / Pod)
Activity Tracking	*Accuracy / Area*	8 bits per fast page (512kB)	16 bits per page (9MB)	N/A	64 MEA entries (736 B)
Migration Trigger	*N/A*	Threshold	Interval	Event	Interval
Tracking Organization	*Simplicity / Parallelization*	Fully centralized (Serialization)	Fully distributed	Fully distributed	Semi-distributed (Pods)
Migration Driver	*Latency*	CPU	CPU (OS)	MCs	Pod
Migration Cost	*Time*	HW cost + CPU	HW cost + SW + TLB + CPU	HW Cost + Communication	HW

Table 1: Breakdown of state-of-the-art designs

Through our evaluation, we identified a good counter size to be 2 bits and 21 bits are needed to address each page within a Pod, leading to a total storage cost of 736B. Using the MEA counters, MemPod's activity tracking profiles *every page in memory* with minimal hardware cost.

MemPod uses timing intervals. At each interval a Pod will migrate up to K pages into its fast memory, where K is the number of MEA counters used. MemPod is transparent to the system, rendering costly OS intervention unnecessary. Since each one of the N Pods will attempt to migrate up to K pages, up to N×K migrations can happen within each interval. However, all Pods can perform their migrations in parallel. Due to MemPod's lightweight activity tracking, intervals can be kept very small, allowing each Pod to better adapt to the application's phase changes. Our evaluation shows a good interval length to be 50us.

With the use of MEA counters, identifying the fast-memory page candidate is as simple as checking that it's not part of the K hot pages. The identification algorithm starts at the very first fast memory location and iterates sequentially until it detects a page address that is not in the set of hottest pages. For the next migration, the identification algorithm simply continues from where it left off. If a hot page already resides in the fast memory it is ignored.

In the state of the art mechanisms presented and evaluated in this paper, building block decisions vary significantly. HMA does not require a remap table due to the OS updating the existing system's structures. For activity tracking it uses Full Counters. The costly OS involvement and the high penalty for sorting all its counters force HMA to operate at very large intervals, weakening its adaptability to phase changes. However, HMA offers full flexibility for migrations.

THM offers significantly limited flexibility by restricting migrations withing segments, however this decision reduces bookkeeping costs significantly. Competing counters in each segment are used for activity tracking, occasionally leading to false (threshold-based) migration triggering if a cold page gets accessed at the right time. Identifying migration candidates incurs very little overhead since there is exactly one fast memory location for each slow memory page that triggers migration.

CAMEO operates similarly to THM, restricting migrations within segments (called congruence groups) but it operates at a finer granularity. Due to its finer granularity, it requires a larger remap table structure than THM. CAMEO does not require activity tracking since it uses an event based migration trigger performing a swap at each slow memory access. The process for identifying migration candidates is identical to THM's.

Table 1 shows a detailed comparison of all mechanisms' building block decisions, comparing their costs and presenting the tradeoff impact of each one.

5.3 Distributed Migration Controllers

Migration mechanisms in the literature [17, 13] assume a centralized migration controller which all memory requests have to go through before reaching the memory, in order to monitor memory activity, read remap tables and control migrations. MemPod's distributed migration controllers control every aspect of the migration process. The use of four Pods breaks the problem into smaller pieces in a divide-and-conquer approach. Instead of trying to identify the N best pages to migrate from the entire memory, each Pod now has to identify the $\frac{N}{4}$ pages per Pod. Each Pod will also require lower bandwidth than a centralized unit as it handles a fraction of the traffic.

Prior publications [29] demonstrate that the layout and address interleaving of main memory and last level cache can be co-designed and benefit a system by increasing efficiency and reducing global traffic. The proposed designs align cache "banks" with main memory banks (among other optimizations). Even though not specifically proposed for 3D-stacked memories, if such a design is assumed, a centralized migration controller would be detrimental to the carefully designed alignment since all LLC misses will now have to go through the centralized unit and then fan back out to their respective MCs.

Finally, a clustered design ensures that we are never moving data across the entire system. Migration can only occur within a Pod and between "sibling" MCs. By limiting migration distance, MemPod imposes a tighter ceiling on data movement energy which can lead to migration-related energy savings when compared to a centralized design.

6. RESULTS

6.1 Evaluation Framework

The goal of our evaluation framework is to quantitatively and qualitatively assess MemPod's capabilities and compare it against state-of-the-art proposed mechanisms. Throughout our evaluation section, we

Processor	
Cores	8 @ 3.2GHz
Pipeline	4 wide out-of-order

Caches	
L1 I-Cache(private)	64KB, 2 way, 4 cycles
L1 D-Cache (private)	16KB, 4 way, 4 cycles
L2 Cache (shared)	8MB, 16 way, 11 cycles

	HBM	DDR4-1600
Capacity	1GB	8GB
Bus Frequency	1 GHz	800 MHz
Bus Width (bits)	128	64
Channels	8	4
Ranks	1	1
Banks	16	16
Row Buffer Size	8KB	8KB
tCAS-tRCD-tRP-tRAS	7-7-7-17	11-11-11-28

Table 2: Experimental framework configuration

	mix1	mix2	mix3	mix4	mix5	mix6	mix7	mix8	mix9	mix10	mix11	mix12
astar		✓					✓	✓	✓✓		✓✓	
bwaves					✓		✓	✓✓	✓			✓
bzip				✓	✓	✓	✓	✓				✓✓
cactus				✓	✓✓	✓		✓				✓✓
dealII				✓✓	✓		✓	✓	✓			✓✓
gcc	✓	✓	✓	✓		✓✓				✓✓		
gems	✓	✓					✓		✓		✓	
lbm	✓		✓			✓				✓		
leslie	✓	✓	✓				✓		✓		✓✓	
libquantum			✓			✓				✓✓		
mcf	✓	✓	✓	✓✓	✓	✓				✓		
milc	✓		✓	✓						✓		
omnetpp	✓	✓						✓			✓	
soplex			✓		✓	✓				✓		
sphinx		✓	✓						✓		✓	
xalanc					✓		✓	✓				✓
zeusmp	✓	✓					✓		✓		✓	

Table 3: Mixed workloads description

study MemPod's performance running with an eight-core CPU. We extend Ramulator [30] to support flat address space hybrid memories. We model the MemPod architecture, as well as HMA, THM, and CAMEO in our simulation framework. Ramulator enables cycle-level memory simulation and includes a simple CPU front-end capable of approximating resource-induced stalls. We evaluate MemPod under a realistic memory configuration consisting of 1GB 3D-stacked HBM [9] and 8GB of off-chip DDR4-1600. Table 2 Provides a more detailed description of the simulated system's configuration.

6.2 Experimental Methodology

We use benchmarks from the SPEC2006 suite [31] as our workloads. Using Sniper [32], we record memory request traces while simultaneously executing 8 benchmarks on a simulated 8-core CPU. We then feed these multi-programmed memory traces into Ramulator, executing all workloads to completion. Our complete set of workloads consists of 15 "homogeneous" workloads, where 8 copies of the same benchmark are run in parallel (we simply refer to these workloads by the benchmark's name in later results), as well as 12 workloads featuring a random mix of 8 benchmarks each (referred to as mix1-12). Each benchmark is executed and traced under its reference input. When running homogeneous workloads, Sniper ensures that memory pages are not shared between workloads. A breakdown of the mixed workloads is shown in Table 3.

We also extended Ramulator with caching for the activity tracking and/or remap tables depending on the simulated mechanism. Bookkeeping-related cache misses inject memory requests into the stream of requests fed by our trace files to retrieve the missing information. No priority is given to these cache miss requests over regular requests. In experiments where the caches for hybrid memory management techniques are disabled, the simulator assumes that any information needed by any mechanism exists on chip and is accessible without any delay. The migration process was implemented in detail as well. In order to read an entire 2KB DRAM page from memory, 32 read requests need to be sent for each of the two migration candidates and then another set of 32 requests for each of the two write-backs.

As we use Ramulator with recorded traces, we report Average Main Memory Access Time (AMMAT) in our results. Even though Ramulator has the ability to approximate IPC with a simple CPU model, AMMAT is computed with much greater fidelity with this tool, as it models the memory system in great detail. AMMAT is the average time spent accessing and waiting for main memory by each request (lower is better).

In our AMMAT experiments, we typically introduce additional accesses to the system (migrations, bookkeeping cache misses). The overhead of the additional misses is accounted to the total memory stall time, but the total memory stall time is divided by the number of original LLC misses (main memory requests) captured in our traces – that is, the denominator in our AMMAT equation does not change between experiments and equals the number of requests in our trace file.

Due to space limitations we are not able to show results for individual workloads in most of the graphs in this paper. In those graphs, we only present the average of all mixed workloads, average of all homogeneous workloads, and overall average.

6.3 Simulation Results

6.3.1 *Page Tracking and Migration Design Space*

MemPod's activity tracking overhead and migration traffic is impacted by the number and size of the MEA counters, as well as the epoch (interval) size over which the counters accumulate. We examine each of these design space parameters in this section. The number of MEA counters dictates the highest possible number of migrations that can be performed at each interval by each Pod, while the epoch length will determine MemPod's ability to better adapt to phase changes in a workload. Furthermore, a small number of counters favors less aggressive migration at each interval boundary. The size of each MEA counter sacrifices accuracy when smaller counters are used but can also save space on the

chip. Smaller counters also sacrifice previous-interval counter accuracy for recency, as it makes it easier to replace a counter for a previously hot page no longer being accessed.

We first identify the optimal number of counters and epoch length by running a series of experiments. We executed all combinations of epoch length and number of counters pairs with epoch lengths of 25-500us. We also exponentially increase the number of MEA counters per pod from 16 to 512. In order to minimize the impact of other factors, we execute this experiment with 16 bits per counter and remap table caches disabled.

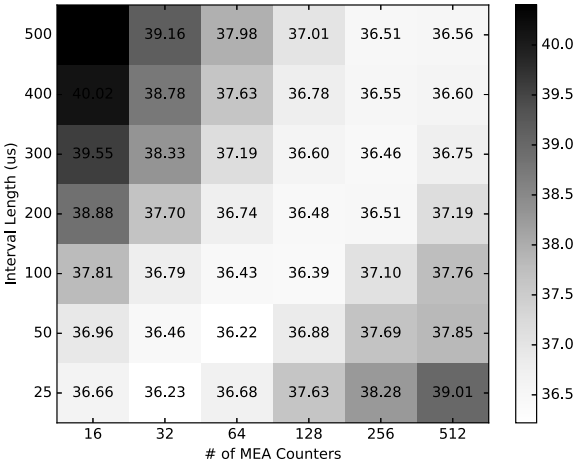


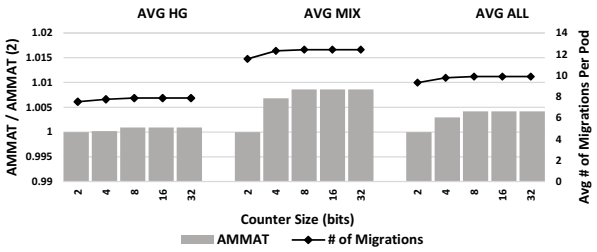
Figure 6: Average AMMAT from all workloads under various MemPod configurations.

Figure 6 shows our results obtained by taking the average AMMAT from all workloads under each MemPod configuration. Based on the results we derive some observations:

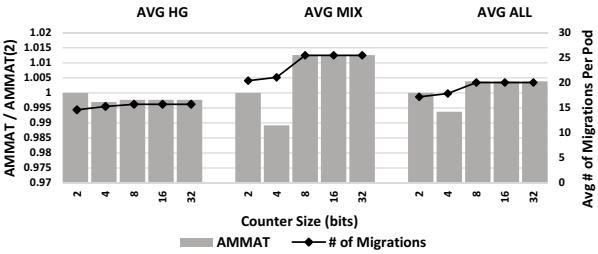
- MemPod achieves the best performance (lower AMMAT) with 50us intervals and 64 counters per Pod. MemPod's lightweight operation allows for such small intervals. For comparison purposes, HMA [14] identified the best epoch length to be 100ms (2000x larger) in order to support all the lengthy processes that take place during a migration event for that method.
- The lowest AMMAT values lie on the diagonal of our result matrix. This implies that the key determinant is the number of migrations, as the maximum migration rate is (very roughly) a constant across the diagonal.
- Few counters and long intervals (inability to react well to phase changes), at least in this sweep of parameters, appears to be worse than many counters and small intervals (overly aggressive migration activity).

The size (in bits) of each counter defines the area requirements of our MEA tracking mechanism. Figure 7a presents AMMAT results normalized to a configuration with 2-bit counters as well as the average number of migrations per Pod per epoch (secondary axis). We first observe that 8 bits are sufficient for our workloads, as larger sizes report identical results. We also observe that reducing the counter size to even less than 8 bits benefits performance, although very marginally. Small counters, as mentioned, favors recency. The smaller the interval, the more important recency becomes over accurate counting; plus, the fewer bits required for accurate counting since there are fewer events. Finally, for 50 us intervals, 2 bit counters offer the best performance (but again, the differences are small).

Figure 7b shows the same experiment with MemPod's parameters set to 100us intervals and 128 counters. This figure demonstrates that as the interval and number of counters increases (i.e. less focus on temporal locality required), the optimal counter size grows from 2 to 4 bits.



(a) MemPod @ 50us, 64 MEA counters



(b) MemPod @ 100us, 128 MEA counters

Figure 7: Counter size (in bits) Vs Normalized AMMAT (primary axis) and average # of Migrations per Pod per interval (secondary axis)

Based on these results, we use 64 MEA 2-bit counters over 50us intervals for subsequent results in this paper. Each one of the 64 MEA entries needs 21 bits for addressing the 1.1M pages per Pod and 2 bits for its counter, leading to an area cost of only 184B per Pod and 736B total. Compared to the state of the art, MemPod's activity tracking requirement is ∼712x smaller than THM's (512KB) and ∼12800x smaller than HMA's (9MB).

6.3.2 *Performance Comparison*

Figure 8 presents a performance comparison of MemPod, HMA, THM, CAMEO and a configuration with 9GBs of on-chip HBM memory, normalized to the performance of a hybrid memory configuration without migration capabilities. We evaluat all mechanisms with migration-related caching disabled.

In order to model HMA's penalties more accurately,

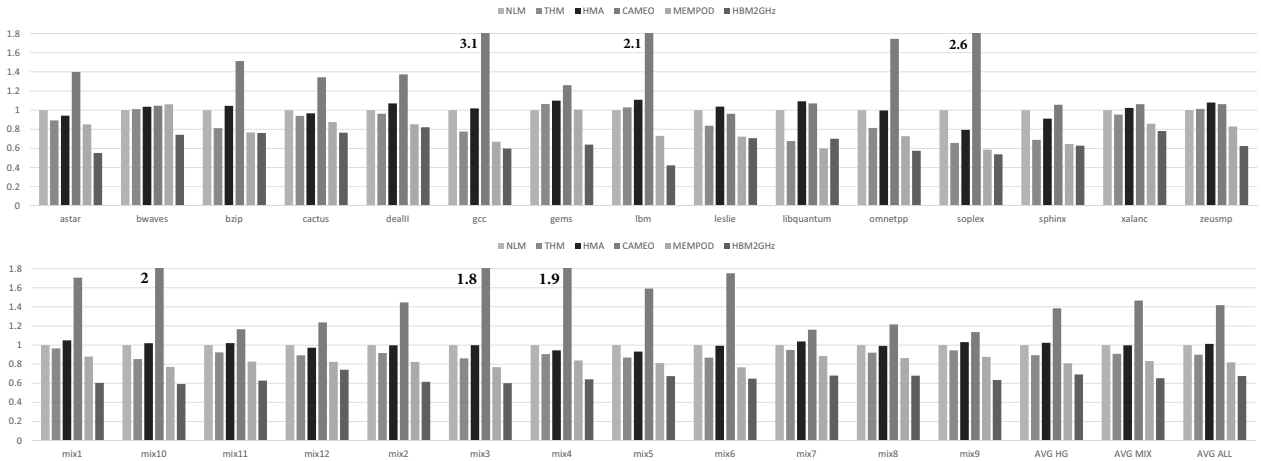


Figure 8: Performance Comparison: AMMAT is normalized to a hybrid memory without any migration mechanism.

we profiled sorting 4.5M integers using quicksort (NlogN complexity) on a real system with a recent Intel Core i7 processor running at 2.1GHz. Our experiment showed that sorting all of HMA's activity tracking counters, at these memory sizes, takes 1.95s on average. When we scale this overhead to our simulated 3.2GHz core, the expected delay would be approximately 1.2s, which is much larger than the proposed optimal interval size for HMA. We instead assumed a very generously reduced overhead of 7ms per interval, assuming we could sort in parallel and discard obvious low values before sorting.

Based on the results we make the following observations:

• MemPod outperforms the state-of-the-art competitors in the majority of our workloads, and in several cases is very close to an HBM-only configuration. MemPod improves AMMAT over a two-level memory without migrations by 19% on average.

• On average, CAMEO reports AMMAT degradation of 41% over the no-migration scheme. The negative impact is caused by our high ratio of slow to fast memory capacity. Given CAMEO's algorithm, 9 slow lines compete for one fast line. At that ratio, it is much more likely for two or more lines to thrash competing for the one spot. From our experiments, we observe CAMEO to force the most movement despite the fact that each move is much smaller. CAMEO moves 3.9GB of data on average per 8-core experiment. For comparison purposes, MemPod moved 3.1GB on average, however migration traffic was divided between Pods, resulting in 804MB per Pod. THM moved 865MB on average and HMA moved 578MB due to its large intervals. We also frequently observe wasted migrations with CAMEO, where a line is evicted before it is touched.

• On average MemPod reports 21% higher AMMAT than HBM-only, while THM and HMA report 33% and 40% respectively.

• In some workloads migration overall is harmful to performance, as observed with the bwaves workload, where a no-migration scheme reports higher performance (lower AMMAT). We observe that in those cases, MemPod leads to deteriorated performance compared to the other mechanisms. However, in the cases of lbm and zeusmp, MemPod increases performance, while THM, HMA and CAMEO report higher AMMAT than the no-migration scheme.

• HMA and MemPod outperform HBM-only when executing the libquantum experiment. In the case of libquantum, the working set size fits entirely in our fast memory. As a result, after some migrations, the entire working set will be present in HBM. With an HBM-only system and no migrations, pages are inserted sequentially by address. In a migration-based system, simultaneously-hot pages are inserted together after each epoch. As the DRAM row buffer is bigger than a page, we find that the co-location of simultaneously-hot pages increases row buffer hit rate from 7% (HBM only) to 90% (MemPod), with 87% of those taking place in fast memory. HMA also sees an improvement in row buffer hit rate. THM and CAMEO cannot take advantage of the small footprint due to their restricted migration flexibility (only one hot page/line per segment can reside in fast memory).

6.3.3 *Caching Effect*

Migration mechanisms will be forced to include a cache as activity tracking and remap table structures are too large to store on-chip. The use of a cache will unavoidably hinder performance. Each mechanism has different cache requirements. THM caches its counters and remap table together with its SRT structure. HMA has no need for a remap table but has high storage requirements for its counting mechanism. MemPod must cache only its remap table as MEA counters easily fit on chip.

In this experiment we evaluate the impact of a cache on MemPod's performance and we also compare it against HMA and THM when operating with a cache. MemPod was configured with the optimal parameter values

identified over the course of this section. Every mechanism was evaluated with 16, 32 and 64 KB of cache. For MemPod, the cache capacity is distributed equally over its four Pods. A part of stacked memory was partitioned to serve as each mechanism's backing store.

In our implementation, each cache miss injects a read request to retrieve missing data. Since all of MemPod's cache misses will occur due to Remap Table updates or lookups it becomes a blocking request for the affected page. All incoming requests to that page need to be delayed until the missing data is retrieved. In-flight requests are not affected by incoming cache misses.

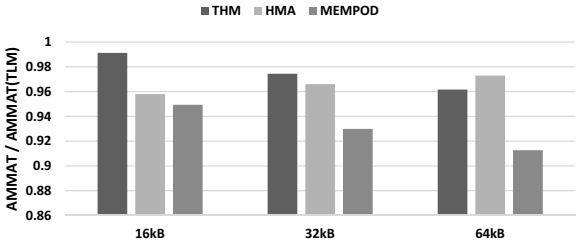


Figure 9: Sensitivity analysis: Cache size. AMMAT normalized to a TLM memory.

Figure 9 shows our results obtained by taking the average AMMAT from all our workloads for each cache size for each mechanism. We present AMMAT results normalized to a two-level memory configuration without any migration mechanism. With 16, 32 and 64kB of cache, MemPod reports 4, 7 and 9% AMMAT improvement over a 2-Level Memory (2LM) with no migration capability and outperforms the other mechanisms.

For 16, 32 and 64kB of cache, the impact on MemPod's performance (when compared to its performance without a cache) is 16, 14 and 12% respectively. THM reports impacts of 12, 10 and 9%. Interestingly, HMA reports lower performance impact with smaller cache sizes rather than larger. Our investigation revealed that the extra cache misses caused by a smaller cache lead to a reduced number of incoming requests to be serviced per interval. As a result, HMA's activity tracking counters have lower values at the end of the interval, leading to fewer migrations. As demonstrated earlier in Section 3, HMA (using Full Counters) has a low prediction accuracy. As such, by reducing the number of migrations, we observe the number of requests serviced by the fast memory to increase, due to hot pages that were not replaced as aggressively.

6.3.4 Scalability

MemPod is designed to be scalable with future technology. If we grow memory sizes by increasing the number of pods, the size of the remap table and the size of the MEA counters will remain constant (per pod, and thus per memory page) if the memory per pod remains constant. If instead we increase memory capacity per pod, the size of the remap table (per memory page) will go up only with the log of the per-pod memory. If we choose to scale the number of counters with the size of memory per pod, it will go up similarly; however, if we do not scale up the counters at the same rate (e.g., four times the memory, but double the counters), then the cost of the counters (per memory page) will go down.

Additionally, the memory traffic caused by migrations will remain distributed and off the primary processor interconnect.

We also expect the latency differential between main memory levels to continue to grow. This will happen as 3D memory parts mature, and as we integrate new memory technologies into the system (e.g. hybrid volatile and non-volatile memory systems). To examine this, we model a system where both the 3D DRAM and the DDR memory are faster, but the 3D DRAM is accelerated further resulting in a higher latency ratio between the two. In particular, we simulate a 4GHz HBM as our stacked memory and a DDR4-2400 as our off-chip memory. Since we are modelling a future system, we reduced the fixed penalty for HMA's sorting process from 7ms to 4.2ms (40% reduction) in order to model future faster processors. We assume no caching effects in this experiment.

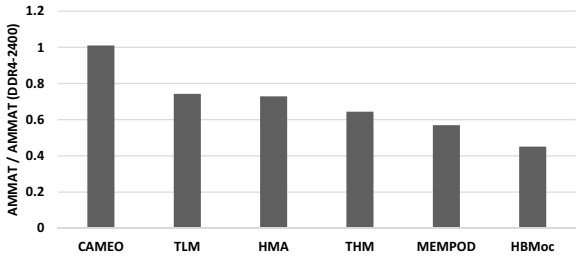


Figure 10: Scalability to faster memories. AMMAT normalized to a DDR4-only memory.

Figure 10 shows our AMMAT results, normalized to a configuration with 9GBs of off-chip DDR4-2400. The label "HBMoc" shows a 9GB configuration with overclocked HBM only. We first observe that CAMEO now reports a 1% AMMAT degradation. The increase in speed differential between stacked and off-chip memory appears to be beneficial for CAMEO, however we can still observe the impacts of intra-segment conflicts. Compared to a configuration with no migration support (TLM), HMA improves AMMAT by 2%, THM by 13% and MemPod by 24%. The overclocked HBM-only configuration is 40% faster compared to TLM.

7. CONCLUSIONS

MemPod is a scalable, modular and efficient dynamic memory management mechanism. Our analysis demonstrates significant gains compared to state-of-the-art proposals. The modular design achieved with the use of Pods allows for a more scalable migration mechanism while at the same time enforcing small limitations on migration opportunities.

MemPod uses MEA counters to track page access activity and identify hot pages. They are dramatically smaller than prior tracking mechanisms while captur-

ing activity counts and temporal recency in a way that provides more effective prediction of future page access.

Acknowledgements

The authors would like to thank the anonymous reviewers for their helpful insights. This research was supported in part by NSF Grant CCF-1302682.

8. REFERENCES

[1] W. A. Wulf and S. A. McKee, "Hitting the Memory Wall: Implications of the Obvious," *Computer Architecture News*, vol. 23, pp. 20–24, March 1995.

[2] JEDEC, "Wide I/O Single Data Rate (Wide I/O SDR)." http://www.jedec.org/standards-documents/docs/jesd229.

[3] Joint Electron Devices Engineering Council, "JEDEC: 3D-ICs." http://www.jedec.org/category/technology-focus-area/3d-ics-0.

[4] J. T. Pawlowski, "Hybrid Memory Cube: Breakthrough DRAM Performance with a Fundamentally Re-Architected DRAM Subsystem," in *Proc. of the 23rd Hot Chips*, (Stanford, CA), August 2011.

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

[6] NVIDIA, "NVIDIA Pascal." http://devblogs.nvidia.com/parallelforall/nvlink-pascal-stacked-memory-feeding-appetite-big-data/.

[7] B. Black, "Die Stacking is Happening," in *Proc. of the Intl. Symp. on Microarchitecture*, (Davis, CA), December 2013.

[8] "AMD Radeon^{TM}R9 series graphics cards with high-bandwidth memory." `http://www.amd.com/en-us/products/graphics/desktop/r9`. Accessed: 2016-04-01.

[9] Joint Electron Devices Engineering Council, "JEDEC: High Bandwidth Memory (HBM) DRAM." https://www.jedec.org/standards-documents/results/jesd235A.

[10] "NVDIMM." `https://www.micron.com/products/dram-modules/nvdimm#/`, 2015. Accessed: 2016-04-01.

[11] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable high performance main memory system using phase-change memory technology," in *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ISCA '09, (New York, NY, USA), pp. 24–33, ACM, 2009.

[12] "PCM." `http://www.extremetech.com/extreme/182096-ibm-demonstrates-next-gen-phase-change-memory-thats-up-to-275-times-faster-than-your-ssd`, 2015. Accessed: 2016-04-01.

[13] 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*, MICRO-47, 2014.

[14] M. R. Meswani, S. Blagodurov, D. Roberts, J. Slice, M. Ignatowski, and G. H. Loh, "Heterogeneous memory architectures: A hw/sw approach for mixing die-stacked and off-package memories.," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, February 2015.

[15] A. J. Chiachen Chou and M. Qureshi, "CAMEO:A Two-Level Memory Organization with Capacity of Main Memory and Flexibility of Hardware-Managed Cache," in *Proc. of the 47th Intl. Symp. on Microarchitecture*, (Cambridge, UK), December 2014.

[16] M. Qureshi and G. H. Loh, "Fundamental Latency Trade-offs in Architecturing DRAM Caches: Outperforming Impractical SRAM-Tags with a Simple and Practical Design," in *Proc. of the 45th Intl. Symp. on Microarchitecture*, (Vancouver, Canada), December 2012.

[17] 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 47th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-47, 2014.

[18] R. M. Karp, S. Shenker, and C. H. Papadimitriou, "A simple algorithm for finding frequent elements in streams and bags," *ACM Transactions on Database Systems (TODS)*, vol. 28, no. 1, pp. 51–55, 2003.

[19] M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams," *Theoretical Computer Science*, vol. 312, no. 1, pp. 3–15, 2004.

[20] S. Volos, J. Picorel, B. Falsafi, and B. Grot, "Bump: Bulk memory access prediction and streaming," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 545–557, IEEE Computer Society, 2014.

[21] V. Karakostas, J. Gandhi, F. Ayar, A. Cristal, M. D. Hill, K. S. McKinley, M. Nemirovsky, M. M. Swift, and O. Unsal, "Redundant memory mappings for fast access to large memories," in *Computer Architecture (ISCA), 2015 ACM/IEEE 42nd Annual International Symposium on*, pp. 66–78, IEEE, 2015.

[22] Y. Du, M. Zhou, B. R. Childers, D. Mossé, and R. Melhem, "Supporting superpages in non-contiguous physical memory," in *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, pp. 223–234, IEEE, 2015.

[23] G. H. Loh and M. D. Hill, "Efficiently enabling conventional block sizes for very large die-stacked dram caches," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 454–464, ACM, 2011.

[24] 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*, pp. 198–210, ACM, 2015.

[25] 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*, pp. 38–50, IEEE, 2014.

[26] P. J. Nair, D. A. Roberts, and M. K. Qureshi, "Citadel: Efficiently protecting stacked memory from large granularity failures," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 51–62, IEEE Computer Society, 2014.

[27] Y. Lee, J. Kim, H. Jang, H. Yang, J. Kim, J. Jeong, and J. W. Lee, "A fully associative, tagless dram cache," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, pp. 211–222, ACM, 2015.

[28] D. Jevdjic, G. H. 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*, pp. 25–37, IEEE, 2014.

[29] G. H. Loh, "3d-stacked memory architectures for multi-core processors," in *Proceedings of the 35th Annual International Symposium on Computer Architecture*, ISCA '08, (Washington, DC, USA), pp. 453–464, IEEE Computer Society, 2008.

[30] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A fast and extensible dram simulator," 2015.

[31] J. L. Henning, "Spec cpu2006 benchmark descriptions," *ACM SIGARCH Computer Architecture News*, vol. 34, no. 4, pp. 1–17, 2006.

[32] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: exploring the level of abstraction for scalable and accurate parallel multi-core simulation," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, p. 52, ACM, 2011.