Page Migration Support for Disaggregated Non-Volatile Memories

Vamsee Reddy Kommareddy
University of Central Florida
Orlando, Florida, USA
vamseereddy8@knights.ucf.edu

Simon David Hammond
Sandia National Labs
New Mexico, USA
sdhammo@sandia.gov

Clayton Hughes
Sandia National Labs
New Mexico, USA
chughes@sandia.gov

Ahmad Samih
Intel Corporation
Austin, Texas, USA
ahmad.a.samih@intel.com

Amro Awad
University of Central Florida
Orlando, Florida, USA
amro.awad@ucf.edu

ABSTRACT

As demands for memory-intensive applications continue to grow, the memory capacity of each computing node is expected to grow at a similar pace. In high-performance computing (HPC) systems, the memory capacity per compute node is decided upon the most demanding application that would likely run on such system, and hence the average capacity per node in future HPC systems is expected to grow significantly. However, since HPC systems run many applications with different capacity demands, a large percentage of the overall memory capacity will likely be underutilized; memory modules can be thought of as private memory for its corresponding computing node. Thus, as HPC systems are moving towards the exascale era, a better utilization of memory is strongly desired. Moreover, upgrading memory system requires significant efforts. Fortunately, disaggregated memory systems promise better utilization by defining regions of global memory, typically referred to as memory blades, which can be accessed by all computing nodes in the system, thus achieving much better utilization.

Disaggregated memory systems are expected to be built using dense, power-efficient memory technologies. Thus, emerging non-volatile memories (NVMs) are placing themselves as the main building blocks for such systems. However, NVMs are slower than DRAM. Therefore, it is expected that each computing node would have a small local memory that is based on either HBM or DRAM, whereas a large shared NVM memory would be accessible by all nodes. Managing such system with global and local memory requires a novel hardware/software co-design to initiate page migration between global and local memory to maximize performance while enabling access to huge shared memory. In this paper we provide support to migrate pages, investigate such memory management aspects and the major system-level aspects that can affect design decisions in disaggregated NVM systems

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
MEMSYS '19, Sept 30 - Oct 3, Washington DC
© 2019 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-9999-9/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

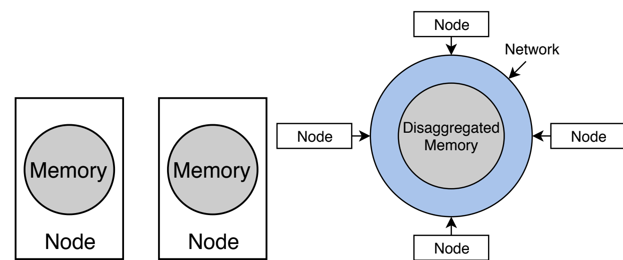
CCS CONCEPTS

• **Computer systems organization → Heterogeneous (hybrid) systems**.

ACM Reference Format:
Vamsee Reddy Kommareddy, Simon David Hammond, Clayton Hughes, Ahmad Samih, and Amro Awad. 2019. Page Migration Support for Disaggregated Non-Volatile Memories. In *MEMSYS '19: The International Symposium on Memory Systems, Sept 30–Oct 03, 2019, Washington DC.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Computing systems with disaggregated memories are being explored as a promising research direction for building future exascale-class computing systems, The Machine project by HP Labs [9, 19]. As many modern applications require large memory allocations in addition to large data sets, scaling memory capacities is becoming a progressively more challenging design requirement. Meanwhile, DRAM technology is proving challenging to scale to a level that copes with the large capacity demands. Moreover, the high idle-power of DRAM requires an expensive cooling infrastructure and excessive operational costs (electric bills). Thus, deploying DRAM in orders of terabytes or petabytes for data centers incurs significant power and cooling costs. Thus, emerging non-volatile memories (NVMs), e.g., Intel's and Micron's 3D X-Point [17], are expected to be the main building block for such systems due to their low idle-power, high speed and scalability.



(a) Conventional systems. (b) Disaggregated memory systems.

Figure 1: Comparison between accessing memory in conventional and disaggregated memory systems.

Disaggregated memory systems differ from conventional computing systems in that memory modules are no longer closely coupled with computing nodes [6, 31], Figure 1. In current High-Performance Computing (HPC) systems, each computing node has a memory module attached to the processor socket directly within the same blade [25], Figure 1a. The major issue of such conventional schemes is that memory modules are only utilized by the co-located computing node, which can leave large percentage of memory underutilized; each node is agnostic to the rest of the system, and thus its memory cannot be utilized efficiently by other nodes [24]. In contrast, disaggregated memory enables fluid division of the shared memory resource among all nodes resulting in a highly scalable system, yet cost efficient, Figure 1b. Finally, upgrading and migrating the memory system would require much less efforts when all memory modules are placed in one central blade, typically called memory blade, as in disaggregated memory systems.

The key enablers for disaggregated memory systems are fast interconnecting interfaces and technologies, e.g., GenZ [11] and CCIX [10], in addition to scalable, dense and ultra-low power memory devices such as emerging NVMs. However, given the contention that results from memory sharing among nodes, proper management of the shared memory resource is a key design requirement. Moreover, since NVM is generally slower than DRAM, it is critical to identify which pages to migrate and when to migrate pages from the shared memory pool to the fast, but small, local memory (typically made out of DRAM). Thus, in this paper we focus on innovating mechanisms that enable efficient memory management scheme in disaggregated memory systems.

Managing disaggregated memory systems is particularly challenging due to the limited number of memory accesses visible to each node. Thus, a centralized scheme in the shared memory resource is more suitable. However, monitoring memory accesses at the shared memory level would require careful design and implementation due to its direct impact on overall system performance. Second, triggering page migrations between shared memory and local memory would require special handling to ensure invalidating the affected and possibly cached memory mappings on each node. Moreover, identifying when and how often to migrate pages from global memory to local memory is challenging due to many aspects: temporal reuse of page, cost of page migration, network latency and shared memory latency. All of these aspects together should be considered to determine if page migration is useful.

Unfortunately, the current literature lacks any detailed study that investigates the system-level aspects and memory management impact on disaggregated memory systems. A large body of research on memory management techniques focuses on a single node with different levels of memory technologies, with advancements in the memory controller for dynamic remapping for hot pages [34]. Other work focuses on hardware/software co-design aspects for enabling monitoring the popularity or "hotness" of each page through TLB structures augmented with counters [30]. Recent work investigates how to manage NVM when used as a swap space, *i.e.*, memory extension [4]. The major difference between prior work and our work is that in disaggregated memory systems, memory management decisions should occur at the *system* level and account for network latency, global memory contentions, global memory latency and the necessary updates of system-level memory mappings. Such considerations pose a challenge on where and how to implement the memory management. For instance, should memory placement be handled by a global memory controller? How aggressively should we perform page migrations? How many pages should be migrated during each epoch, and, what page migration costs would still render page migration useful? While the objective is the same in any heterogeneous memory system work: *migrate hot pages to the faster memory*, the design aspects and usefulness of page migration strictly depend on the system architecture, memory architecture and memory technologies. Thus, designing and implementing memory management in disaggregated memory systems has its own unique challenges, conclusions and design guidelines. Moreover, most of the prior work either conclude their results based on trace-driven or analytical models [3, 30], or use real-system profiling where additional latency is added on each page fault [4], and, hence, are inapplicable to systems where global memory is directly accessible, i.e., not like a swap device. In contrast, in this paper, we use and provide a detailed cycle-level simulation model, which has been integrated with a best-of-class open-source simulation framework and is able to replicate the significant detail associated with the major system-level aspects that affect memory-management decisions.

In this paper, we systematically analyze the impact of various memory management aspects including TLB shootdowns, page migration latencies, page migration frequency and initiation mechanisms, and global memory latency, on the overall system performance for several applications. We identify what system configurations and parameters would make page migration more useful. Finally, we propose a novel page migration mechanism that relies on minimal hardware changes to track hot pages at the global memory, where such information can be periodically accessed by system software to initiate page migrations at defined epoch boundaries.

To evaluate our proposed memory management scheme, we use the Structural Simulation Toolkit (SST), a cycle-level architectural simulator that is widely used, based on open-source licensing and is publicly available. We use SST to model in detail page migration overhead, TLB shootdown latency, shared external interconnects, and global NVM memory. We simulate a prototype configuration of eight computing nodes that share a global memory. While disaggregated memory systems are expected to have hundreds of computing nodes, given the shared memory resource, simulating such large system can lead to many serialized events and hence will produce intractable simulation times. However, since memory-centric systems are likely to have groups of memory modules connected through high-speed memory networks, our simulation model captures instances where the computing node to memory module ratio would still be maintained throughout the system, e.g., 128 nodes would share 16 memory modules (8:1 ratio). Therefore, the same conclusions and insights would still apply. The only exception is for applications that actually share large percentages of data in the global memory, e.g., data analytics, where network contentions and hot spots can largely affect system performance. However, since our focus in this work is on HPC applications, where data sharing is limited, our conclusions still apply. Simulating hundreds (or maybe thousands) of nodes with a shared memory is likely impractical with the current state of detailed simulation models and beyond

the scope of this paper although we hope to return to extensions in this area in future work.

Our evaluation results reveal that for a system with unoptimized TLB shootdown costs, page migration latency, and memory-centric latency, applications do not benefits from page migration, since migrating pages to the relatively fast local memories is amortized by the costs of page migration and TLB shootdown. To mitigate this, we first investigate the impact of optimized TLB shootdown latency to understand to what level TLB shootdown cost is acceptable in disaggregated memory systems, i.e., the point on which such migrations no longer burden memory management. Later, we vary page migration latency and global memory latency to understand their impact on the effectiveness of memory management. Based on these investigations, we propose a novel mechanism that removes much of these overheads from the critical path. In summary, this paper makes the following contributions:

- We propose and discuss a novel memory management framework to enable direct-access for disaggregated memory systems. We discuss several hardware/software co-design aspects to enable our support.
- We thoroughly analyze the different system-level aspects that affect memory management decisions, and identify the bottlenecks and future opportunities.
- We thoroughly analyze counter-intuitive observations and discuss the pitfalls of common assumptions about memory management in disaggregated memory systems.
- We provide to the public a cycle-level simulation model that facilitates exploring memory management aspects in disaggregated memory systems.

The remainder of the paper is organized as follows: Section 2 presents the background; Section 3 and Section 4 discuss the design and evaluation of our implementation and we conclude the paper in Section 5.

2 BACKGROUND AND MOTIVATION

Since the topic of this paper focuses on disaggregated memory systems, in this section we briefly define our approach to building them. Later, we describe the process of page migration, a common memory management operation in heterogeneous memory systems. Finally, we discuss and motivate our paper by discussing the importance of analyzing disaggreageted memory systems.

2.1 Disaggregated Memory Systems

Disaggregated memory systems are increasingly perceived as one of the most suitable design options for future HPC systems. In disaggregated memory systems, compute resources are decoupled from memory modules, i.e., some of the memory modules are removed from each compute node and then grouped together in memory blade(s) on a high-performance system-scale interconnect. Many users typically refer this fabric attached memory as a kind of global memory available to compute nodes running their application. As such, theoretically, compute nodes can access any memory location in such global/centralized memory blade(s), which enables efficient data sharing and better utilization of memory capacity. Moreover, such decoupling of memory modules from memory nodes, and relying on standard interconnect interfaces instead, can enable more flexible integration of heterogeneous compute nodes and units, e.g., accelerators and specialized compute units. In addition, new compute resources can be more flexibly added or removed as an organization's workloads grow and flow over time.

Since disaggregated memory systems are most suitable for large memory capacities, dense non-volatile memories (NVMs) are generally assumed as the main building blocks of such memory systems. Unlike conventional DRAM, NVMs do not require frequent refresh operations, and hence have ultra-low idle power requirements [12, 13, 20, 27, 29, 39]. Moreover, NVMs enable persistent storage of files and data, and thus are more applicable for systems with a high degree of data sharing as in data analytics. Generally speaking, disaggregated memory systems are expected to have both global memory (which can be based on relatively slow NVM), and much smaller local memory (based on DRAM) which is coupled with each compute node.

While the benefits of disaggregated memory systems are typically presumed as directly applicable to HPC systems, we are not aware of any study on the impact of such a design when used in HPC environment. Features such as flexible upgrade, better utilization and easier system integration are generally obvious, however, the impact of memory contentions and memory management in such systems are unclear. In particular, some of the most pressing questions are when and how to migrate pages from and to global memory, and, when to perform TLB shootdown operations which is a costly process that is required to maintain consistency of cached page table entries.. Therefore, this paper serves as a detailed study that demonstrates the effects of such system aspects and thoroughly investigates novel mechanisms and memory management techniques to further improve the performance.

2.2 Page Migration

Moving frequently accessed pages from global memory to local memory would benefit both node and system overall performance, due to the reduced contention at the global memory and the fast response time of local memory. However, as system-level management deals with each node's physical page mappings, it is an obvious task for system-level management to dynamically migrate hot pages to the local memories accessing them more frequently. While naively we can assume such pages can be identified at the start time of application and then migrated to local memory, the reality is that many challenges and application behaviors typically render such simple solutions ineffective. In particular, the dynamic nature and time-variable pages popularity (access frequency) render static solutions irrelevant. Moreover, given the limited capacity of local memory, an accurate and dynamic profiling of page behavior is needed, otherwise more popular pages may get evicted from local memory.

2.3 Translation Look-Aside Buffer (TLB) Shootdown:

TLBs are fast caches that hold page table translations within each core. However, TLB coherence with the full system page table must be maintained explicitly through inter-core interrupts and explicit invalidation commands. When performing page migration, processor cores have to be stalled to invalidate page table entries in

respective TLB structures [1, 2]. The process of clearing stable page entries is called TLB shootdown [3] and can become an expensive operation.

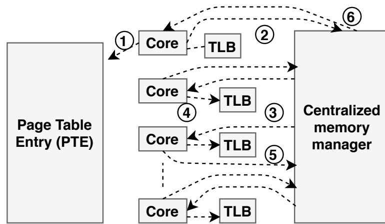


Figure 2: TLB shootdown process

Figure 2 shows the process of TLB shootdown in disaggregated memory systems. The core, which invalidates the addresses, will update the page table entry (PTE) (1) and send a TLB shootdown request to the centralized memory manager (2). Centralized memory manager stalls all the other cores (3) and the stalled cores invalidate the copies of page table entry in their respective TLBs (4). After invalidating the TLB entries, the stalled cores send an acknowledgement request to the centralized memory manager and resume (5). When the centralized memory manager receives shootdown acknowledgements from all the cores, it resumes the shootdown of the initiating core (6). The entire TLB shootdown procedure consumes around 8us, on an average, in 8 core computing system [36].

2.4 Motivation

To understand the impact of disaggregated memory systems on performance, we need to analyze instructions per cycle (IPC) when varying the number of compute nodes. Since it is expected to have a fixed compute node to memory module ratio, we vary such ratio to understand the severeness of possible contentions and the design space ranges that are possible. Note that within each memory blade, it is expected to deploy very fast on-node networks between groups of memory groups, and, as such memory networks are likely to feature relatively high on-node bandwidths. However, since our focus on this paper aims to understand the contention for specific memory sharing levels, we vary the number of nodes sharing global memory modules, selecting from 1, 2, 4 and 8 nodes per memory module. Such ratios reflect different workloads and allocation policies of a larger systems; a specific module in a larger system would likely have multiple nodes that access it frequently, however, a system memory allocation policy would try to limit such number of sharers by dividing global memory into memory pools where each pool corresponds to specific nodes.

Figure 3 shows performance of individual nodes in disaggregated memory systems and traditional systems, with NVM as memory for a range of HPC-relevant benchmarks (see Section 4 for additional discussion). As the number of nodes accessing global memory increase, the performance of the applications decreases due to more contention at global memory. We consider performance when global memory is not shared among multiple nodes (Disag-NVM-N1) as the optimal case (no contention) in disaggregated systems. As expected, the performance worsens when the number of nodes sharing global memory increases, e.g., 4 and 8 nodes. For instance, for a memory intensive workload like Lulesh, we can notice a decrease in the performance by 83.8% with 8 nodes normalized to no contention case. Meanwhile, some other workloads, which are less memory sensitive, e.g., NAS IS, encounter 39% slowdown when moving from 1 node to 8 nodes sharing the same memory module. As shown in Figure 3, the memory response time increases differently for each workload, mainly due to the variable levels of memory request rate and contention. Since memory

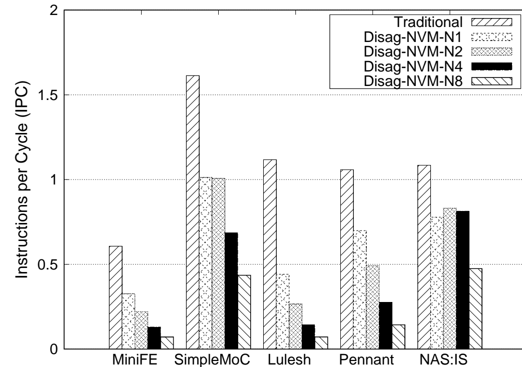


Figure 3: Delay per request in accessing memory for traditional and disaggregated memory systems

access delays can be significant when multiple nodes access the global memory concurrently, effectively reducing such delays is a key design requirement for disaggregated memory systems. To reduce the memory access latency, pages can be migrated to the local memory when necessary, which can effectively reduce the contention as well. In this paper, we devise and analyze a memory management support which relies on dynamic page profiling and migration between global memory and local memory, and hence effectively improve overall system performance.

3 MEMORY MANAGEMENT SUPPORT

In this section, we discuss our proposed memory management support for disaggregated NVM memory systems. Our scheme relies on page migration as a mechanism to enable more efficient page locality and data proximity to their most-accessing compute nodes. To design our memory management support, we start with identifying the answers for the following questions: (1) Which pages to migrate to local memory? (2) Which pages to select as victim pages, i.e., be evicted from local memory?

3.1 Detecting Hot and Victim Pages

Pages that needs to be migrated between the two levels of memory should be chosen carefully. Wrong selection of pages would degrade the performance of the application intensely. For instance, if a page in the local memory is accessed frequently and is migrated to the global memory, during the page migration process, the number of cycles to fetch the data of that page would be more. Apart from increasing the number of cycles to access frequently accessed data, a number of cycles would be wasted due to TLB shootdown. Hence

efficient page selection algorithms to migrate pages are required to improve the overall performance of the system. For detecting hot pages, we leverage a counter-based scheme, however, we use clock-based replacement policy to select victim pages, as discussed below.

Page Insertion: Page insertion techniques are used to detect hot pages in the global memory and migrate them to the local memory. We leverage on counter-based scheme to select the pages to insert in the local memory since counter based scheme is simple and accurate. In this scheme every page access is accounted and during the process of page migration, the page counters are traversed to find out the most frequently accessed pages. Page accesses are stored in page access count table (PACT) as shown in Figure 4. This policy, as it seems, is simple, but there are two important overheads that should be considered. *Hardware Requirements:* As each page requires a separate counter, the number of counters needed would be more for systems that has significant amount of memory, specifically in disaggregated memory systems. This needs tremendous amount of additional hardware. A solution to overcome this is to maintain a cache of counters in the global memory controller and a counter is fetched from the memory during the cache miss. *Traversal Delay:* The page counters, either maintained as hardware counters or as a cache has to be traversed to find out the most and least frequently accessed pages. The number of entries in PACT will be significantly high for the systems with huge memory and a moderate page size (4KB). If the number of entries in PACT is huge, it takes a while to traverse the table to find out the most frequently accessed pages. Although page accounting and page selection process for migration can be performed at the background, while the data is fetched from the memory, eventually it is a costly operation to traverse PACT. We address these two concerns by fixing the PACT size and replacing the least frequently accessed page with in PACT to make space for the new page and storing the least frequently accessed page counter data in the memory. This eliminates huge hardware requirements and reduces the delay in traversing PACT.

In disaggregated memory systems, multiple nodes access the global memory. Hence global memory should have the ability to distinguish requests from different nodes. The memory controller achieves this by extracting the node number from the request packet. In Figure 4, *B* is termed as hot page detector. Hot page detection is performed in three steps: 1) During serving a request the page number is extracted from the base address and the page count is incremented in PACT if the page entry is found. If the page entry is not found in PACT, an entry is created and is then incremented. 2) If the page counter is higher than the page migration threshold limit the page is copied to the *pages to be migrated table per node* (PMTn). Global memory controller has to maintain PMT for every node since each node will have different pages to migrate.

Page Eviction: Page eviction technique is used to detect victim pages to be migrated to the global memory from the local memory. Counter based eviction policy is widely used as a page replacement policy to migrate pages between main memory and the secondary memory. Least recently used pages are selected and are moved to the secondary memory. We extend this scheme for selecting least recently used local pages (victim pages) to be migrated to the global memory. In clock based page selection policy each page is referenced, *A* in Figure 4. But unlike counter based method, this policy maintains per page reference bit in the page table rather than a counter which requires 32 or 64 bits per page. If the reference bit of the page is set then it is considered as a page which is accessed recently and vise verse. Initially all the page references are reset and for every page access the reference bit is set to indicate that it is accessed recently. A reference pointer is utilized to traverse the page table to select the victim pages by verifying the page reference bit. The reference pointer traverses the page reference table until it finds a page that is not accessed recently. While traversing the page table, the page reference which is set is reset by the reference pointer and the traversal continues. To lessen the delay in finding out the victim page the page table is traversed until specific entries (200 in our case) and once it is reached, the victim page is chosen as the page pointed out by the reference pointer by default. Selecting victim pages is triggered by victim page eviction handler, *D* from Figure 4, during page migration epoch.

3.2 Performing Page Migration:

Centralized memory manager is required to maintain and allocate decoupled centralized memory. We used Opal [23] from SST[35] as a centralized memory manager. Opal is responsible for allocating memory to all the nodes without any conflicts. We extend Opal to perform page migration. For every *page_migration_epoch*, Opal communicates with the global memory controller and individual nodes to fetch pages that needs to migrate. Hot page insertion handler, *C* in Figure 4, fetches addresses of hot pages from the global memory controller. The global memory controller which already segregated pages to be migrated during hot page detection, sends the respective page addresses (page numbers), if any, to Opal after sorting the PMTn table to migrate the most frequently accessed pages first and then clears PACT and PMT to collect page counts for the next epoch interval. Victim page eviction handler, *D* from Figure 4, fetches local memory pages to be moved to the global memory, with the help of clock based eviction method. Once both hot and victim pages are fetched by Opal, TLB invalidation and TLB shootdown events along with pages addresses to be remapped are sent to the respective nodes involved in page migration, *E* from Figure 4. Nodes which does not have any pages to migrate are not interrupted. The page contents are swapped by Direct Memory Access (DMA), *F* from Figure 4, during TLB shootdown.

3.3 Sequence of Events:

The sequence of events are as follows: a) For every memory request pages are referenced at either the local node or at the global memory controller. If the memory request is to the local memory then the respective page is referenced in the page table of the local memory manager (A from Figure 4). If the memory request is to the global memory then the page access is counted at the global memory controller and hot pages are detected and stored in PMTn (*B* from Figure 4). b) Centralized memory manager, Opal, for every specific time interval, for instance 1M clock cycles, triggers the global memory and individual nodes to fetch hot and cold pages (*C* and *D* from Figure 4). c) Once Opal receives page addresses that has to be migrated, it triggers DMA to swap pages between global and local memory while TLB shootdown and TLB invalidation

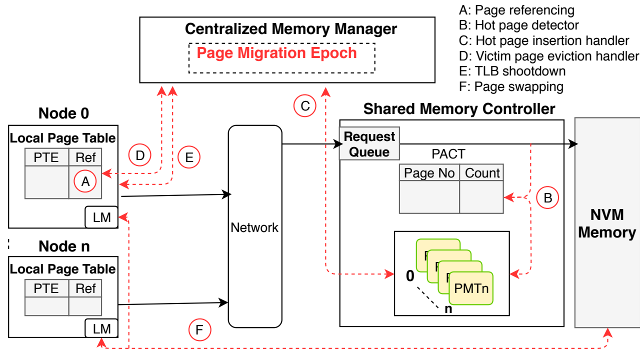


Figure 4: Page migration in disaggregated memory systems. LM: Local Memory, PMTn: Pages to Migrate Table per node, PACT: Page Access Count Table

events are initiated to only those nodes which are involved in page migration (*E* and *F* from Figure 4).

3.4 Discussion:

Page migration does not always better the system. This depends on factors like frequency of page migration, number of pages migrating at a time, page migration threshold, TLB shootdown latency and page swapping delay.

It is also crucial to interpret at what frequency the page migration process should be performed. If the page migration is performed too often then most of the cycles would be wasted in migrating pages and TLB shootdowns. If the page migration is performed rarely then the advantage in migrating pages is lost since most frequently accessed pages would be migrated to local memory rarely during the lifetime of the application. Conventionally, various page migration schemes for different architectures talk about migrating one page during page migration process, while it is possible to migrate multiple pages at a time. The delay in migrating multiple pages is less compared to migrating single page. For instance, if we consider a $1us$ delay to swap contents of one page and if we assume a delay of $8us$ to invalidate TLB entries and update page table entry, the total delay to migrate one page is $9us$. If pages are migrated one at a time then every time a single page is migrated, the computing units have to wait for $9us$. If more pages are migrated at a time then only the page swapping delay ($1us$) per page will be added, which is effective. Page migration threshold is an other factor which has an impact on pages to migrate. If the migration threshold is less then the pages with very less accesses to global memory would be eligible to migrate. Also if the migration threshold is pretty high then the pages with very high accesses to global memory would not be eligible to migrate, which results in less page migrations. During TLB shootdown and page swapping, computing units are stalled and are not allowed to proceed util all the pages are swapped and all the TLB levels are invalidated. Hence it must be understood that it is vital to study the impact of these factors on page migration in disaggregated memory systems.

3.5 Overhead:

There are mainly three overheads associated with our design: 1) Hardware overhead: Global memory controller requires additional hardware for PACT and PMT. This overhead is minimal since PACT and PMT are fixed based on the number of pages to migrate. If we assume 100 pages to migrate at a time then each PMT (PMTn) should have a minimum of 100 entries and PACT should store a minimum of 800 entries, considering 8 nodes system. 2) Accounting overhead: Each page has to be accounted at the global memory whenever the page is accessed, which is in the critical path. This can be avoided from the critical path by performing such accounting in the background. 3) Page address transfer overhead: During the page migration, metadata like page addresses, are exchanged between centralized memory manager and global memory controller or local memory management units. This overhead is minimal since statistical data transfer would happen only during the page migration epoch and if the page migration epoch interval is high then the statistical data transfer would be minimum.

4 EVALUATION

To study page migration aspects in disaggregated memory, we used a model of a disaggregated system developed in the Structural Simulation Toolkit (SST) [35]. SST has been proven to be one of the most reliable simulators for large-scale systems due to the scalability and modular design of its components. SST includes multiple (swappable) simulation modules for various components. A module called *Opal* [23] has been developed in SST to simulate centralized memory manager for disaggregated memory model.

Table 1: Simulation Parameters

Element	Parameters
CPU	2 Out-of-Order cores, 2GHz, 2 issues/cycles, 32 max. outstanding requests
L1	private, 64B blocks, 32KB, LRU
L2	private, 64B blocks, 256KB, LRU
L3	shared, 64B blocks ,16MB, LRU
Local memory	256MB, DDR4-based DRAM
Global memory	16GB, NVM-based DIMM (PCM), 128 max. outstanding requests, 16 banks $300ns$ Read Latency, $1000ns$ Write Latency
External network latency	40ns

Table 2: Applications

Application	Value
Lulesh [22]	-s 120
SimpleMoC [15]	-t 2 -s
Pennant [14]	leblancbig.pnt
miniFE [18]	-nx 140 -ny 140 -nz 140
NAS:IS [5]	class C

We simulated disaggregated memory system with 8 nodes. All the nodes run simultaneously with each node hosting a benchmark. Simulation parameters of our simulation environment are shown in Table 1. According to the table, 2 cores are used for each node and each core can serve up to 2 instructions per cycle. The clock

frequency of the cores is 2GHz, with each core configured to serve up to 100 million instructions of application execution during its HPC-relevant kernels. Three levels of cache are used, L1, L2, and L3, with sizes 32KB, 256KB and 16MB respectively and cache type is non-inclusive. Local memory is 256MB of DRAM memory on each node. Centralized memory is of an NVM type and configured to be 16GB, based on density compared to DRAM and number of nodes. Network latency is critical in disaggregated memory system. External network latency is 40*ns*, which has been modelled after public projections for a GenZ-enabled network.

Since our focus is on HPC applications we evaluated our design using 5 HPC-relevant mini-applications and benchmarks. Lulesh [22], a mini-app for unstructured hydrodynamics, Pennant [14] is an unstructured mesh physics mini-app designed for advanced architecture research, SimpleMoC [15] is a mini-app to demonstrate the performance characteristics and viability of the Method of Characteristics (MOC) in 3D neutron transport calculations in the context of full scale light water reactor simulation. The IS benchmark from the NASA Parallel Benchmark collection [5] is an integer sort kernel which performs efficient large-scale sorting operations. Finally, MiniFE [18] is a proxy application for unstructured implicit finite element codes. Applications along with their parameters are shown in Table 2. We decided upon these specific applications as these are known to provide memory accesses that are characteristic of larger codes. In most cases, their access patterns are memory-access intensive although the range of these accesses which is satisfied by caching, or accesses to memory, will vary by each kernel. Note that for our discussion of the following experiments, N indicates number of compute nodes.

4.1 Effect of Page Migration Parameters

We modelled a combination of a threshold based method and clock based approach to migrate pages from between centralized and private memory. As mentioned in Section 3, performance gains of using page migration is dependent on migration frequency (page migration epoch), the number of pages to migrate at a time and page migration threshold to detect a page requiring migration. Hence we vary these parameters and study the effect of them. Figure 5 shows normalized performance (IPC) results with respect to disaggregated memory system without page migration. The x-axis of each subgraph indicates PMT at global memory which decides if the page is a hot page or not. For instance, an x-axis of 50 in each graph indicates that when a shared page is accessed more than 50 times in the current epoch, then that specific page is marked as a page to be migrated to the private memory and might get migrated to the local memory during the migration interval. We varied PMT from 10 to 100 accesses. Graphs in each row show results for a specific migration epoch. For example, the migration interval of graphs in each column of row 1 is 10K cycles. While rows 2, 3, 4 and 5 indicate results for migration epoch of 100K, 1M, 10M, 100M cycles respectively. Each column represents the maximum pages that can be migrated. For instance, results in column 1 are configured to migrate a maximum of 1 page for every migration epoch and columns 2, 3, 4 and 5 migrate a maximum of 10, 50, 100, 500 pages per epoch respectively. The top most frequently accessed pages are chosen if the number of pages to be migrated exceeded maximum pages to be migrated per epoch.

We make the following observations by varying these parameters from Figure 5. If PMT is high, pages should be accessed frequently to count them as hot pages. On the other side if PMT is low, most of the pages would be counted as hot pages and there would be a pool of hot pages to choose from for migrating to local memory. Hence PMT decides the hotness of the pages. Therefore we divide our explanation into two parts: (1) High PMT (Pages are counted as hot pages if the frequency of page accesses is very high), and, (2) Low PMT (Pages are counted as hot pages if they are accessed less number of times).

High PMT: If PMT is high, a page should be accessed frequently to be counted as a hot page. In our results we observed that when the epoch size is small, 10K and 100K cycles (row 1 and 2), all the applications that we simulated, do not improve the performance when the page migration threshold is set to 50 accesses or beyond. This is because when the epoch length is small, the number of global memory pages getting accessed more than 50 times in that epoch is very narrow and hence none of the pages would reach the required PMT, resulting in no page migration. Hence when the epoch size is small, 50 accesses is defined as high PMT irrespective of the number of maximum pages to migrate. When the epoch size is larger – 1M and 10M cycles – MiniFE benefits from page migration even if the PMT is beyond 50 accesses (1.46x, 1.33x, 1.45x and 1.44x for 10, 50, 100 and 500 maximum pages to migrate respectively with a migration epoch of 1M cycles). And the performance is normalized to 1 when PMT is set at 100 accesses. Therefore for MiniFE, a high PMT is defined as 50 accesses if epoch size is less and 100 accesses if epoch size is more.

Low PMT: If PMT is low, most of the pages, if accessed regularly, are eligible to be counted as hot pages, hence many hot pages would be moved to local memory and improve the performance. It can be seen that apart from when only 1 page is allowed to migrate per epoch and when epoch interval is too high (100M cycles), we can observe performance improvement. If only one page is allowed to migrate per epoch, the cost of page migration, explained in 4.2, outweighs the benefits. Also if page migration epoch interval is too high, page migration frequency is very low which leads to no page migration. Hence there is no improvement in performance (row 5). There is improvement if the number of pages to migrate is 500, since even though the frequency is less, there are a higher number of pages able to be migrated. Although migrating 500 pages at a time is not practical, or at least is likely to present a significant migration cost, we show these results to see the variation in performance improvement. When the migration epoch size is less, 10T and 100T cycles, the benefits of page migration is nullified by page migration cost (row 1). Hence there is no improvement in performance for MiniFE. But for SimpleMoC and Pennant applications, the improvement is around 1.4x, 1.3x for 10, 50, 100 and 500 pages to migrate at maximum per epoch and is around 1.85x and 1.58x more for 10, 50, 100 and 500 pages to migrate at maximum per epoch with PMT of 10 and migration epoch of 10T and 100T cycles. For applications like NAS-IS the impact of page migration cost severely affects the performance. Performance degrades by 30% and 28% when PMT is at 10 and 30 accesses. As the epoch size increases from 10T to 100M the effect of page migration costs decreases as the

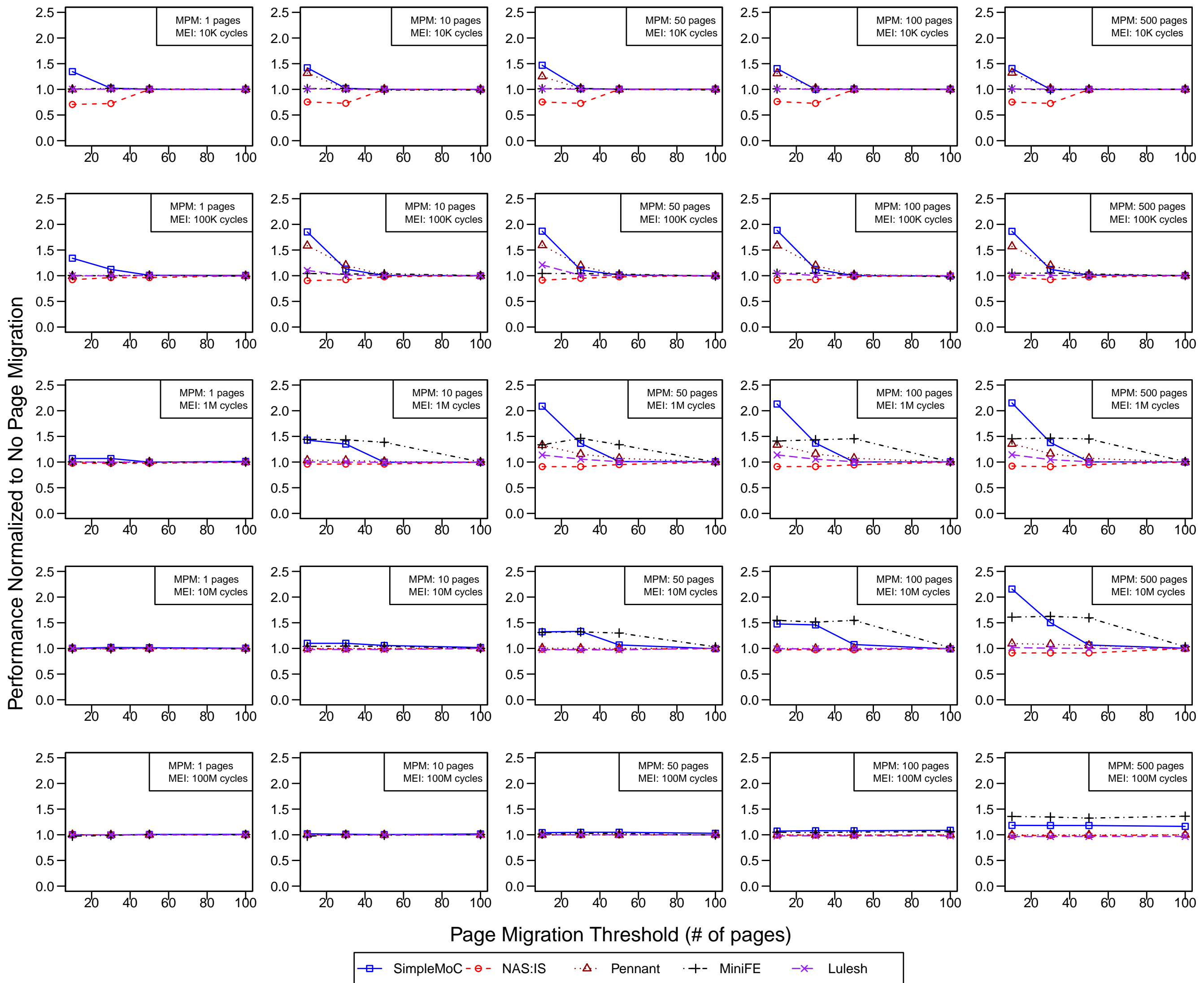


Figure 5: Performance improvement (normalized to no page migration) in disaggregated memory system when page migration parameters migration epoch length, number of pages to migrate and page migration threshold are varied. Shootdown latency is maintained at 8us and per page migration delay(cost) is 1us. MPM indicates Maximum pages to Migrate per epoch. MEI indicates Migration Epoch Interval

frequency at which page migrations are performed is less. Hence the performance of the NAS-IS application is normalized to 1. As the epoch size is more 1M and 10M cycles the performance gain for SimpleMoC and Pennant applications diminishes. SimpleMoC achieves peak performance improvement of 2.08x, when migration epoch is 1M cycles and a maximum of 50 pages to migrate with a PMT of 10. MiniFE also benefits from page migration if the migration epoch is maintained at 1M and 10M cycles (1.4x more with PMT of 10, 30 and 50). The peak performance improvement can be noted when epoch interval is 10M cycles and a maximum of 500 pages to migrate with a PMT of either 10, 30 or 50 accesses (1.6x) but migrating 500 pages at once is not advisable. Due to space constraints, moving forward, we will only show the best possible cases for all the applications- PMT of 10 (low), 50 pages to migrate and the migration epoch interval of 10K, 100K and 1M cycles.

4.2 Page Migrations Costs

From the above observations it should be understood that intensive page migration leads to severe page migration costs. Page migration costs can be classified into three categories: (a) *Invalidating TLB units - TLB shootdown latency:* TLB shootdown latency can be

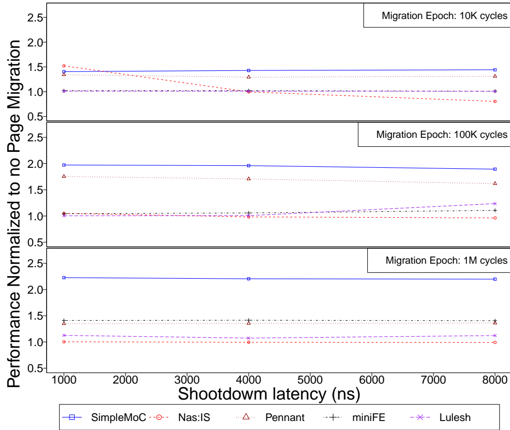


Figure 6: Performance improvement (normalized to no page migration) in disaggregated memory system with best possible cases. Migration epoch interval is 10T, 100T and 1M cycles with PMT of 10 accesses, 8us TLB shootdown latency, a maximum of 50 pages to migrate per epoch and per page migration delay(cost) of 1us per page.

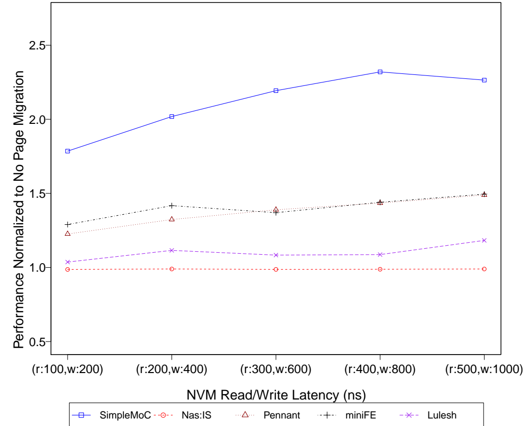


Figure 7: Performance improvement in disaggregated memory system with respect to conventional memory system with different NVM read/write latency. PMT is 10 accesses with a maximum of 50 pages to migrate, TLB shootdown latency of 8us and per page migration delay(cost) is 1us per page. Page migration is performed for every 1M cycles.

reduced by using schemes like self-invalidating TLB's [3]. (b) *Swapping page content delay:* While the pages are undergoing swapping, computing units might operate on yet to be swapped page. Since a batch of pages are undergoing swapping computing units which operate on these pages should halt. We call this page swapping delay. Usually pages are swapped with the help of DMA engines. To account for page swapping operation, we add an additional 1us latency in our experiments. Accordingly, page swapping cost increases by 1us for every page that gets migrated from central memory to local memory. If there are 50 pages to migrate then page swapping cost is 50us. This can be reduced by intuitively recording pending pages that will undergo swapping and performing page swapping atomically. That is, MMU units of each node decides if the address translation should proceed or not by checking if the page associated with the address is marked as a pending pages to swap. If the page is not marked, MMU would proceed with the address translation. If the page is marked and if the status of the page indicates that the page is undergoing swapping, the MMU waits till the swapping is done. If the page is not undergoing swapping but in the pending pages to be swapped list, MMU would proceed with the addresses translation without waiting. With this approach and with the help of atomic page swapping, page swapping delay can be completely nullified. (c) *Cost to find pages to migrate:* This is minimal since most of the work has been done while accounting for the page accesses in the background.

Leveraging on reducing page migration cost schemes, we intuitively evaluated page migration in disaggregated memory systems by varying TLB shootdown latency. Figure 6 shows the results for the best case configuration according to Figure 5, column with maximum number of page migrations at a time is 50 (column 3) and a PMT of 10 accesses, with varying TLB shootdown latency (8us to 1us) as shown on the x-axis. We optimistically choose a batch of 50 page to migrate at a time and nullified page swapping cost, since we believe that during TLB shootdown, DMA engine would have swapped 50 pages between global and local memory. With low TLB shootdown cost (1us), applications like NAS:IS whose performance was degrading with TLB shootdown cost of 8us, 0.8x, could improve its performance by 1.52x (with migration epoch of 10K). As the epoch size increases, the improvement due to page migration is reduced. 1.05x and 1.0x for NAS:IS application when TLB shootdwon delay is 1us with 100K and 1M migration epoch intervals. For Lulesh application the improvement is 1.75x, 1.7x and 1.61x for 1us, 4us and 8us TLB shootdown delay respectively. For other applications the improvement is marginal.

4.3 Sensitivity to NVM's Read/Write Latency

Centralized memory in disaggregated memory systems must meet several requirements. One of them is the ability to allocate memory to all nodes accessing it. NVM memory technology is a perfect candidate to fullfil such requirement as the density of NVM is much higher than DRAM [16]. On the other hand, NVMs are notorious for the high write latency compared to DRAM. And although that NVM's read latency is much better than its write latency counterpart, it is still slower than the read latency of a typical DDR generation. We believe that read/write latency are crucial in the migration of pages to/from centralized memory. To this end, we studied page migration benefits by varying NVM read/write latency. We categorize NVM into 5 categories - very fast (read and write latency is 100 and 200ns), fast (read and write latency is 200 and 400ns), moderate (read and write latency is 300 and 600ns), slow (read and write latency is 400 and 800ns) and very slow (read and write latency is 500 and 1000ns).

We can intuitively expect that if the global memory is "fast", then baseline scheme would perform just well without any page migration. On the other hand, if the global memory is "slow", we expect page migration to optimize performance significantly versus

the baseline of no page migration, as we reduce the number of "slow" memory accesses. To showcase this, and since the improvement due to page migration is evident for all applications when pages are migrated every 1M cycles, we use this as our migration epoch.

In Figure 7, 'r' indicates read latency and 'w' indicates write latency. For example (r:100,w:200) indicates NVM read latency of 100ns and write latency of 200ns. As the type of the NVM varies from very fast to very slow, the benefits of page migration is more clear. For instance performance gains due to page migration for SimpleMoC with very fast global memory is 1.78x and with very slow global memory the performance gain is 2.3x. For MiniFE and Pennant the improvement is around 1.25x to 1.48x when the global memory is varied from very fast to very slow. The improvement due to page migration for Lulesh application with very fast NVM as global memory is hardly 3%, however, when the global memory is very slow the performance gain reaches up to 18%

5 RELATED WORK

A large body of prior art has investigated page migration in hybrid memory systems which proposes efficient way to migrate pages. Ramos et al. [34] proposed a multi-queue based approach to define the hotness and coldness of the pages. Wang [38] managed NVM at a super-page granularity by using a lightweight page migration. Wang also considered the utility of the migrating page. Yoon et al. [40] devised a policy that enables DRAM to cache pages which has high frequency of row buffer misses in NVM memory. CAMEO [8], PoM [37], Mempod [33] and BATMAN [7] discuses about the granularity and relaxations possible while swapping pages to maximize overall memory bandwidth. Other approaches [21, 28] involve both hardware and software. OS is utilized to identify hotness of the page. Page migration is explored in NUMA architectures [32]. These approaches depend on either compiler support or Linux kernel and leverage on counters for number of pages accesses. Lim et al. [26] proposed software-based prototype by extending the Xen hypervisor to emulate a disaggregated memory design wherein remote pages are swapped with local memory on-demand upon access, first touch policy. They also explored round-robin, clock and content based page placement policies to effectively manage the memory. Specifically content-based approach can be effectively utilized for page sharing.

Most of the previous schemes perform page swapping at a predefined time intervals and does not take time interval variation into consideration. Also Lim et al. [25, 26] implemented disaggregated memory design on Xen hypervisor. In this paper, we study and analyze different factors that impact the page migration benefits in disaggregeted memory model using cycle-based simulation environment, SST. We introduce a new memory management scheme that leverages a combination of replacement policies, clock and counter based, to perform page migration. We understand that the page migration benefits are dependent on factors like PMT, migration interval, maximum number of pages to migrate per epoch, TLB shootdown and page swap delay. We vary these factors and study the effect of these parameters on the overall performance of the system. Also, we found that the type of memory used as global memory has an effect on page migration. Hence we analyzed page migration aspects under different global memory speeds.

6 CONCLUSION

Disaggregated memory systems is a promising future architecture which has the ability to impact future systems design. These memory architectures provide a path to solve current memory concerns in scalability, with the sharing of huge data sets such as social graphs, as well as complex scientific data-sets in HPC. The challenge associated with their design in an HPC context, is how to best utilize the resources of the system and balance these against the ever increasing demands to achieve better performance.

To this end, we have proposed a novel memory management scheme for disaggregated memory systems. As disaggregated memory systems support both local and shared memory, we identify hot pages in global memory and provide migration capabilities to local memory using a combination of threshold based and clock based policies. We provide insights into the impact of migrating pages at regular intervals, showing that the benefit of page migration is dependent on factors like page migration epoch, the maximum of number of pages to migrate at each epoch, whether a page migration threshold can be used to differentiate hot and cold pages as well as the costs associated with TLB shootdowns and page swapping delays. We evaluated 5 HPC-relevant applications and benchmarks to study the effect of these factors on page migration. We showed that Non-volatile Memory (NVM) is a feasible memory type to construct disaggregated memory model, and hence we studied the effect of NVM read/write latency on page migration in disaggregated memory systems. We show the best case improvement of up to 2.3x when page migration is applied on a disaggregated memory system with slow NVM as main memory.

7 ACKNOWLEDGMENTS

This work has been funded through Sandia National Labora-tories (Contract Number 1844457) Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International,Inc., for the U.S. Department of EnergyâĂŹs National Nuclear Security Administration under contract DE-NA0003525.

REFERENCES

[1] Nadav Amit. 2017. Optimizing the TLB shootdown algorithm with page access tracking. In *Proc. USENIX Ann. Conf.* 27–39.

[2] A. Arpaci-Dusseau. 2000. Translation Lookaside Buffers (TLBs). http://pages.cs.wisc.edu/~eli/537/lectures/TLB.2x2.pdf

[3] Amro Awad, Arkaprava Basu, Sergey Blagodurov, Yan Solihin, and Gabriel H Loh. 2017. Avoiding TLB Shootdowns Through Self-Invalidating TLB Entries. In *Parallel Architectures and Compilation Techniques (PACT), 2017 26th International Conference on.* IEEE, 273–287.

[4] Amro Awad, Sergey Blagodurov, and Yan Solihin. 2016. Write-aware management of NVM-based memory extensions. In *Proceedings of the 2016 International Conference on Supercomputing*. ACM, 9.

[5] David H Bailey. 2011. Nas parallel benchmarks. In *Encyclopedia of Parallel Computing*. Springer, 1254–1259.

[6] Daniel Turull Chakri Padala and Vinay Yadav. 2017. Time for memory disaggregation? Ericsson Research Blog. *Online]. https://www.ericsson.com/research-blog/time-memory-disaggregation/* (may 2017).

[7] Chiachen Chou, Aamer Jaleel, and Moinuddin Qureshi. 2017. BATMAN: techniques for maximizing system bandwidth of memory systems with stacked-DRAM. In *Proceedings of the International Symposium on Memory Systems.* ACM, 268–280.

[8] Chiachen Chou, Aamer Jaleel, and Moinuddin K Qureshi. 2014. 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.* IEEE Computer Society, 1–12.

[9] Dan Comperchio and Jason Stevens. 2014. Emerging Computing Technologies: Hewlett-PackardâĂŹs âĂIJThe MachineâĂİ Project. In *HP Discover 2014 conference held in Las Vegas June 10-12*. Willdan Energy Solutions, 1–4.

[10] CCIX Consortium. 2017. *Online]. https://www.ccixconsortium.com/* (2017).

[11] GenZ Consortium. 2017. GenZ Core Specification. *Online]. https://www.ericsson.com/research-blog/time-memory-disaggregation/* (May 2017).

[12] Howard David, Chris Fallin, Eugene Gorbatov, Ulf R Hanebutte, and Onur Mutlu. 2011. Memory power management via dynamic voltage/frequency scaling. In *Proceedings of the 8th ACM international conference on Autonomic computing*. ACM, 31–40.

[13] Qingyuan Deng, David Meisner, Luiz Ramos, Thomas F Wenisch, and Ricardo Bianchini. 2011. Memscale: active low-power modes for main memory. In *ACM SIGPLAN Notices*, Vol. 46. ACM, 225–238.

[14] Charles R Ferenbaugh. 2015. PENNANT: an unstructured mesh mini-app for advanced architecture research. *Concurrency and Computation: Practice and Experience* 27, 17 (2015), 4555–4572.

[15] Geoffrey Gunow, John Tramm, Benoit Forget, Kord Smith, and Tim He. 2015. Simplemoc-a performance abstraction for 3d moc. (2015).

[16] J. Cao H.-Y. Chen S. B. Eryilmaz S. W. Fong J. A. Incorvia Z. Jiang H. Li C. Neumann K. Okabe S. Qin J. Sohn Y. Wu S. Yu X. Zheng H.-S. P. Wong, C. Ahn. [n. d.]. Stanford Memory Trends. Retrieved February 1, 2019 from https://nano.stanford.edu/stanford-memory-trends

[17] Jim Handy. 2015. Understanding the Intel/Micron 3D XPoint memory. In *Proc. SDC*.

[18] Michael A Heroux, Douglas W Doerfler, Paul S Crozier, James M Willenbring, H Carter Edwards, Alan Williams, Mahesh Rajan, Eric R Keiter, Heidi K Thornquist, and Robert W Numrich. 2009. Improving performance via mini-applications. *Sandia National Laboratories, Tech. Rep. SAND2009-5574* 3 (2009).

[19] Forbes Technology Council Jai Menon. 2018. The Rise Of Memory-Centric Architectures. *Online]. https://www.forbes.com/sites/forbestechcouncil/2018/11/16/the-rise-of-memory-centric-architectures/* (November 2018).

[20] Brian G Johnson and Charles H Dennison. 2004. Phase change memory. US Patent 6,791,102.

[21] Sudarsun Kannan, Ada Gavrilovska, Vishal Gupta, and Karsten Schwan. 2017. Heteroos: Os design for heterogeneous memory management in datacenter. In *ACM SIGARCH Computer Architecture News*, Vol. 45. ACM, 521–534.

[22] Ian Karlin, Jeff Keasler, and JR Neely. 2013. *Lulesh 2.0 updates and changes*. Technical Report. Lawrence Livermore National Laboratory (LLNL), Livermore, CA.

[23] VR Kommareddy, A Awad, C Hughes, and SD Hammond. [n. d.]. Opal: A Centralized Memory Manager for Investigating Disaggregated Memory Systems. ([n. d.]).

[24] Shuang Liang, Ranjit Noronha, and Dhabaleswar K Panda. 2005. Swapping to remote memory over infiniband: An approach using a high performance network block device. In *2005 IEEE International Conference on Cluster Computing*. IEEE, 1–10.

[25] Kevin Lim, Jichuan Chang, Trevor Mudge, Parthasarathy Ranganathan, Steven K Reinhardt, and Thomas F Wenisch. 2009. Disaggregated memory for expansion and sharing in blade servers. In *ACM SIGARCH Computer Architecture News*, Vol. 37. ACM, 267–278.

[26] Kevin Lim, Yoshio Turner, Jose Renato Santos, Alvin AuYoung, Jichuan Chang, Parthasarathy Ranganathan, and Thomas F Wenisch. 2012. System-level implications of disaggregated memory. In *High Performance Computer Architecture (HPCA), 2012 IEEE 18th International Symposium on*. IEEE, 1–12.

[27] Chung-Hsiang Lin, Chia-Lin Yang, and Ku-Jei King. 2009. PPT: joint performance/power/thermal management of DRAM memory for multi-core systems. In *Proceedings of the 2009 ACM/IEEE international symposium on Low power electronics and design*. ACM, 93–98.

[28] Felix Xiaozhu Lin and Xu Liu. 2016. Memif: Towards programming heterogeneous memory asynchronously. *ACM SIGARCH Computer Architecture News* 44, 2 (2016), 369–383.

[29] Song Liu, Brian Leung, Alexander Neckar, Seda Ogrenci Memik, Gokhan Memik, and Nikos Hardavellas. 2011. Hardware/software techniques for DRAM thermal management. (2011).

[30] Mitesh R Meswani, Sergey Blagodurov, David Roberts, John Slice, Mike Ignatowski, and Gabriel H Loh. 2015. 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*. IEEE, 126–136.

[31] Hugo Meyer, Jose Carlos Sancho, Josue V Quiroga, Ferad Zyulkyarov, Damian Roca, and Mario Nemirovsky. 2017. Disaggregated computing. an evaluation of current trends for datacentres. *Procedia Computer Science* 108 (2017), 685–694.

[32] Guilherme Piccoli, Henrique N Santos, Raphael E Rodrigues, Christiane Pousa, Edson Borin, and Fernando M Quintão Pereira. 2014. Compiler support for selective page migration in NUMA architectures. In *Proceedings of the 23rd international conference on Parallel architectures and compilation*. ACM, 369–380.

[33] Andreas Prodromou, Mitesh Meswani, Nuwan Jayasena, Gabriel Loh, and Dean M Tullsen. 2017. MemPod: A clustered architecture for efficient and scalable migration in flat address space multi-level memories. In *High Performance Computer Architecture (HPCA), 2017 IEEE International Symposium on*. IEEE, 433–444.

[34] Luiz E Ramos, Eugene Gorbatov, and Ricardo Bianchini. 2011. Page placement in hybrid memory systems. In *Proceedings of the international conference on Supercomputing*. ACM, 85–95.

[35] Arun F Rodrigues, K Scott Hemmert, Brian W Barrett, Chad Kersey, Ron Oldfield, Marlo Weston, Rolf Risen, Jeanine Cook, Paul Rosenfeld, E CooperBalls, et al. 2011. The structural simulation toolkit. *ACM SIGMETRICS Performance Evaluation Review* 38, 4 (2011), 37–42.

[36] Bogdan F Romanescu, Alvin R Lebeck, Daniel J Sorin, and Anne Bracy. 2010. UNified instruction/translation/data (UNITD) coherence: One protocol to rule them all. In *Proceedings-International Symposium on High-Performance Computer Architecture*.

[37] Jaewoong Sim, Alaa R Alameldeen, Zeshan Chishti, Chris Wilkerson, and Hyesoon Kim. 2014. Transparent hardware management of stacked dram as part of memory. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 13–24.

[38] Xiaoyuan Wang. 2018. Supporting Superpages and Lightweight Page Migration in Hybrid Memory Systems. *arXiv preprint arXiv:1806.00776* (2018).

[39] H-S Philip Wong, Simone Raoux, SangBum Kim, Jiale Liang, John P Reifenberg, Bipin Rajendran, Mehdi Asheghi, and Kenneth E Goodson. 2010. Phase change memory. *Proc. IEEE* 98, 12 (2010), 2201–2227.

[40] HanBin Yoon, Justin Meza, Rachata Ausavarungnirun, Rachael A Harding, and Onur Mutlu. 2012. Row buffer locality aware caching policies for hybrid memories. In *Computer Design (ICCD), 2012 IEEE 30th International Conference on*. IEEE, 337–344.