CHAMELEON: A Dynamically Reconfigurable Heterogeneous Memory System

Jagadish B. Kotra[1,2], Haibo Zhang[2], Alaa R. Alameldeen[3], Chris Wilkerson[4], Mahmut T. Kandemir[2]
[1]AMD Research (Work done while at The Pennsylvania State University), Email: Jagadish.Kotra@amd.com
[2]The Pennsylvania State University, Email: {haibo, kandemir}@cse.psu.edu
[3]Intel Labs, Email: alaa.r.alameldeen@intel.com
[4]NVIDIA, Email: cwilkerson@nvidia.com

***Abstract*—Modern computing systems and applications have growing demand for memories with higher bandwidth. This demand can be alleviated using fast, large on-die or die-stacked memories. They are typically used with traditional DRAM as part of a heterogeneous memory system and used either as a DRAM cache or as a hardware- or OS-managed part of memory (PoM). Caches adapt rapidly to application needs and typically provide higher performance but reduce the total OS-visible memory capacity. PoM architectures increase the total OS-visible memory capacity but exhibit additional overheads due to swapping large blocks of data between fast and slow memory.**

In this paper, we propose Chameleon, a hybrid architecture that bridges the gap between cache and PoM architectures. When applications need a large memory, Chameleon uses both fast and slow memories as PoM, maximizing the available space for the application. When the application's footprint is smaller than the total physical memory capacity, Chameleon opportunistically uses free space in the system as a hardware-managed cache. Chameleon is a hardware-software co-designed system where the OS notifies the hardware of pages that are allocated or freed, and hardware decides on switching memory regions between PoM- and cache-modes dynamically. Based on our evaluation of multi-programmed workloads on a system with 4GB fast memory and 20GB slow memory, Chameleon improves the average performance by 11.6% over PoM and 24.2% over a latency-optimized cache.

I. Introduction

Many client, mobile, and data-center applications have a growing demand for memories with higher bandwidth and larger capacity. With the increasing popularity of many data-intensive applications including high-resolution graphics and machine learning, memory is increasingly becoming a performance bottleneck. Due to the significant speed disparity between CPUs and memory, improvements in memory latency and bandwidth can lead to substantial performance improvements in memory-bound applications. The advent of die-stacked or on-die memories (e.g., HBM [1], HMC [2]) proved to be instrumental in bridging this performance gap.

Unfortunately, the high bandwidth memories cannot be used as a sole memory component in a system due to limited number of stacks impacting the capacity and the interposer challenges. This observation led to heterogeneous memory systems[1] that combine a fast memory component (e.g., die-stacked DRAM) with a (typically larger) relatively slow memory components (e.g., DDR3/DDR4) [3–6]. Many prior proposals use fast memory in a heterogeneous memory system as a cache [7–16]. Caches provide performance advantages [17–21] due to spatial and temporal locality where a large fraction of memory references access fast memory. However, since caches duplicate data, they reduce the overall Operating Systems (OS)-visible memory capacity which degrades performance for high memory-footprint applications. This is especially a problem when fast memory constitutes a large fraction of the overall memory capacity.

To enable the performance improvements of fast memory without losing capacity to caches, Part of Memory (PoM) architectures which expose the stacked DRAM to the OS to enhance the overall OS-visible capacity have been proposed [22–25]. In a datacenter environment, exposing the additional capacity of stacked DRAM to OS is beneficial since:

- It enables the datacenter schedulers to schedule more jobs, reducing the overall waiting time of the jobs, thereby improving datacenter throughput.
- It reduces the number of page faults for some pathological scenarios not foreseen by the system administrator or the user who specified constraints while submitting a job.
- It can reduce system cost and power by reducing the overall amount of off-chip DRAM required (e.g., a system with 4GB/16GB stacked/off-chip DRAM could be replaced by a system with 4GB/12GB stacked/off-chip DRAM for a total of 16GB OS-visible memory).

PoM architectures could be hardware-managed [22, 25] or could be used as an OS-managed NUMA system [23, 24]. OS-managed memory systems provide a low-overhead mechanism to achieve performance, but do not react quickly to changing memory demands of running applications. Hardware-managed PoM systems adapt quickly to changing memory demands and therefore outperform OS-managed heterogeneous memory, but this comes at the expense of higher area and power to manage hardware address indirection and large region swaps between fast and slow memories [25]. Due to the necessity of swapping large segments or pages between the fast and slow memories, PoM could degrade performance when swaps[2] interfere with

[1]Note that in this paper, a heterogeneous memory system refers to a system that includes memories having varying bandwidths. Memories referred to as "fast" have *higher bandwidth* (and not latency) compared to the "slow" ones.

[2]In this paper, a swap refers to a segment swapped between stacked and off-chip DRAMs, while the page swap between an off-chip DRAM and a secondary storage device like SSD or disk is implied by a page-fault.

on-demand memory accesses.

In this paper, we propose and evaluate a novel architecture, Chameleon, which attempts to achieve the best of cache and PoM architectures using hardware-software co-design. Chameleon uses a hardware-managed PoM as the baseline to achieve its capacity advantages. However, we rely on the OS to inform hardware of any pages that have been allocated or freed using two new instructions: *ISA-Alloc* and *ISA-Free*. Based on this information from the OS, Chameleon opportunistically uses the freed pages as a hardware-managed cache, therefore avoiding the expensive page swap operations. Chameleon switches between cache and PoM modes dynamically based on the available free space. More specifically, this paper makes the following contributions:

- We show using real system experiments that different workloads exhibit different memory-footprints over time.
- We propose novel Chameleon and Chameleon-Opt co-designs. Our basic Chameleon design leverage the free space in stacked DRAM as cache, while an optimized Chameleon-Opt remaps the stacked DRAM segments to off-chip proactively to free space in stacked DRAM to be used as cache. Our proposed Chameleon co-designs provide PoM-like memory capacity with cache-like performance.
- We propose simple changes in the instruction set architecture to support Chameleon. With two new instructions, the OS can inform the hardware when a page is freed or allocated, allowing Chameleon to use free pages as a cache.
- Our simulation results show that Chameleon performs better than static PoM and cache architectures, outperforming a PoM baseline by 11.6% and a latency-optimized cache by 24.2%.

II. Background and Related Work

A. NUMA Dichotomy

1) Multi-Socket Homogeneous Memory: In a traditional multi-socket system, each socket consists of on-chip cores, a last-level cache (LLC), and a local off-chip DRAM as depicted in Figure 1a. Sockets are connected to each other through an interconnect, e.g., AMD Hyperconnect. For tasks running on Socket-0, accessing data from local DRAM connected to socket-0 incurs lower latency compared to accessing data from the remote memory connected to socket-1. The extra latency incurred in the remote access is due to an additional interconnect traversal latency, even though the local and remote memories have the same bandwidth. This variation in access latencies between the local and remote memories results in a Non-Uniform Memory Access, widely referred to as NUMA. This NUMA dichotomy leads to computation/data placement having a significant impact on performance.

2) Single-Socket Heterogeneous Memory: With the advent of high-bandwidth stacked DRAMs integrated on the die through a silicon transposer, each socket is turning into a NUMA system. This is because accesses to the on-chip stacked DRAM are faster compared to off-chip memory. Figure 1b shows a block diagram of a single-socket heterogeneous system containing stacked and off-chip DRAMs.

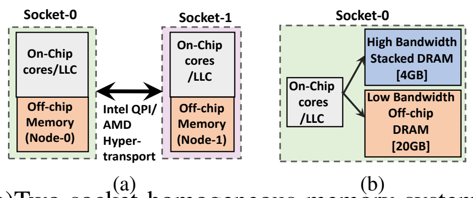


Fig. 1: (a)Two-socket homogeneous memory system, and (b) Single-socket heterogeneous memory system

B. NUMA-Aware OS Optimizations

1) NUMA-Aware Memory Allocator: Traditional NUMA-Aware Linux and VMware ESXi, by default, allocate each task's data in the same socket on which the task is running to maximize local memory accesses [26–28]. This is widely referred to as "first-touch" or local memory allocation policy in Linux. These allocation policies improve performance by minimizing remote memory accesses.

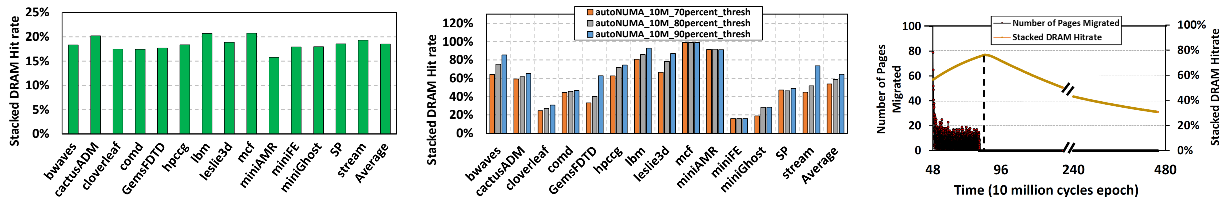
2) Linux Automatic NUMA Balancing (AutoNUMA): Linux supports an advanced Automatic NUMA balancing mechanism to enhance the locality between the executing task and its corresponding memory [29]. On a multi-socket system, AutoNUMA keeps track of local-to-remote memory accesses by poisoning some set of pages (i.e., invalidating the corresponding page table entries). As a result, a processor load/store to the corresponding page results in a page-fault. In a given epoch, referred to as "numa_balancing_scan_period" in AutoNUMA, Linux calculates the "remote-to-local page-fault ratio". At the end of numa_balancing_scan_period epoch, if the remote-to-local page-fault ratio exceeds a threshold (referred to as "numa_period_threshold"), the misplaced pages which caused remote page-faults are migrated from the remote socket's memory to the local socket's memory. Depending on the remote-to-local page-fault ratio, the numa_balancing_scan_period is updated dynamically so that the misplaced pages can be migrated quickly to the local socket.

One important issue in AutoNUMA is that memory pages are migrated from the remote to local socket only as long as there is enough free space available in the local socket's memory. If there is no free space left, page migration fails with "-ENOMEM" error. If the remote-to-local fault ratio continues to increase, since AutoNUMA can no longer migrate memory pages to local socket, it migrates the task from a current socket (say socket-0) to socket-1 to minimize remote memory accesses.

C. Hardware-Managed Heterogeneous Memory

In the context of single-socket heterogeneous memory systems, the hardware-managed techniques for stacked DRAM primarily falls into three categories: (1) cache-based systems and (2) PoM-based systems, and (3) statically reconfigurable hybrid memory systems.

1) Stacked DRAM as Cache: A large body of recent work has looked at utilizing stacked DRAM as another cache between the last level cache (LLC) and system memory [9–14, 16, 30]. A DRAM cache provides good performance and software transparency, but needs to appropriately organize the cache structure (data and tags). To study the architectural implications



(a) (b) (c)

Fig. 2: (a) NUMA-Aware Allocator (b) AutoNUMA Hit rates, (c) Cloverleaf AutoNUMA timeline (for 90% threshold).

of DRAM cache, prior proposals looked at direct-mapped [14], set-associative [13, 16] and fully-associative [12] cache designs.

2) Stacked DRAM as Part of Memory (PoM): Recent works also studied the usage of stacked DRAM as an OS-visible extension to off-chip memory [22, 25, 31–33]. In particular, [31, 32] proposed software-hardware approaches that warrant OS to detect and collect page access information by identifying the first requested pages and the hot pages (FTHP). While [33] uses a Majority Element Algorithm (MEA) algorithm originally proposed for databases to track and predict hot-pages to migrate to the stacked DRAM, [34] uses a locking-based sub-blocked architecture to swap blocks between stacked and off-chip DRAMs to maximize the overall bandwidth. However, proposals [22, 25] explored hardware-based redirection via a hardware remapping table to ensure high stacked DRAM hit rates. While the design in [25], optimized for lower meta-data overhead and spatial locality by storing frequently accessed 2KB segments in stacked DRAM, CAMEO [22] employed a 64-Byte segments trading off bandwidth for high meta-data overhead and spatial locality. Further details on the design of [22, 25] are covered in Section V. All these proposals [22, 25, 31–34] are agnostic to the OS-visible free space in managing the heterogeneous memory system.

3) Statically Reconfigurable Heterogeneous Memories: KNL supports various modes [35] of stacked DRAM (referred to as MC-DRAM) operation. These modes include: (1) 100% cache, (2) 100% OS-visible flat (memory), and (3) static hybrid-mode. Apart from these two modes, KNL also supports two static hybrid-mode configurations. In one configuration, 25% of stacked DRAM operates in cache-mode while the remaining 75% operates in memory-mode. In the second configuration, 50% of stacked DRAM operates in cache-mode while the remaining 50% operates in memory-mode. These modes are static and require a system reboot to switch from one hybrid configuration to another.

III. Challenges in Architecting a Performance-Optimized Heterogeneous Memory

A. OS-based NUMA-Aware Solutions

To understand how existing OS-based NUMA-aware solutions fare in a single-socket heterogeneous memory system, we used a full-system GEM5 [36] simulator setup (more details of experimental setup in Section VI-A) running Linux OS kernel-4.2.2. The kernel is compiled with "CONFIG_NUMA=y" and "CONFIG_NUMA_EMU=y" options to emulate NUMA setup. However, since the existing kernel does not support emulating asymmetric capacities for the NUMA nodes, the Linux kernel was modified to support different capacities with a patch similar to [37] and is configured using "numa=fake=1*4096,1*20480". As a result, the Linux kernel now emulates two NUMA nodes, one with 4GB capacity and another with 20GB capacity on a system with only one physical node. Further, to model the NUMA effects due to bandwidth variation between stacked- and off-chip DRAMs, we modified the memory controller module in GEM5 to simulate heterogeneous memory bandwidths.

1) NUMA-Aware Memory Allocator: The Linux NUMA-aware "First-touch" allocator (Section II-B1, [38]) tries to allocate as many pages as possible in the faster, stacked DRAM to increase the stacked DRAM hit rate. Figure 2a shows the stacked DRAM hit rate for high memory footprint workloads in a system containing 4GB stacked DRAM and 20GB off-chip DRAM. The average stacked DRAM hit rate for these high footprint workloads is as low as 18.5%. This low stacked DRAM hit rate is due to two main factors. First, the non-proportional capacity of the stacked DRAM compared to the off-chip DRAM limits the amount of data that a stacked DRAM can accommodate (4GB in our experiments) before it runs out of memory. Second, the OS lacks adequate hot-page prediction mechanisms. Employed page allocation strategies can result in some of the hot-pages getting allocated in the off-chip DRAM, reducing stacked DRAM hit rate. Our results demonstrate that the NUMA-Aware first-touch memory allocator policy is not optimal for heterogeneous memory systems as it results in severe under-utilization (low hit rate) of the stacked DRAM.

2) Linux AutoNUMA: Some of the shortcomings mentioned in Section III-A1, are successfully handled by Linux AutoNUMA [29]. Figure 2b shows the stacked DRAM hit rate for a 4GB stacked + 20GB off-chip DRAM system for different numa_period_threshold values (70%, 80% and 90%). The higher numa_period_threshold yields better stacked DRAM hit rates, an average of 64.4%, as the misplaced (off-chip DRAM) pages are migrated more rapidly to the stacked DRAM. Though the average hit rate of AutoNUMA is better compared to the NUMA-Aware allocator, the hit rates are still undesirably low. Specifically, Cloverleaf has a cumulative hit rate as low as 30.7% (for 90% threshold). The timeline graph in Figure 2c helps us reason about the lower hit rate in the Cloverleaf workload. The primary Y-axis represents the number of pages migrated per epoch, while the secondary Y-axis shows the stacked DRAM hit rate[3]. The X-axis represents the timeline where each epoch is 10 million processor cycles. With time, as the number of misplaced pages migrated from off-chip

[3]Note that there are already pre-allocated memory pages in the stacked DRAM as it is exposed to OS in AutoNUMA.

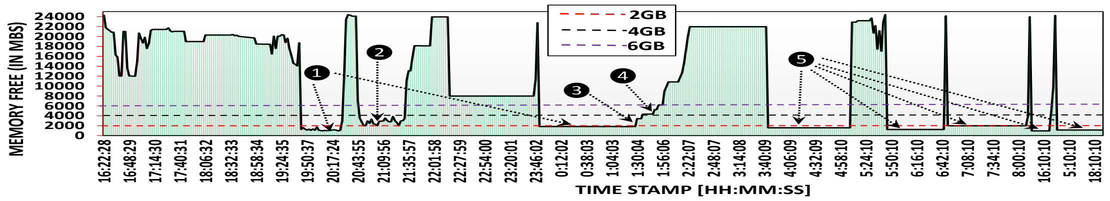


Fig. 3: Inter-Workload Memory Footprint Variation Over Time [16:22:28 (day-1) - 22:10:10 (day-3), Elapsed Time: 53.8 hours].

DRAM to the stacked DRAM increases, the stacked DRAM hit rate increases, reaching a maximum of around 77.1% at the 81st epoch. However, after epoch 81, the stacked DRAM hit rate reduces gradually from 77.1% to 30.7%. This is because as pages are migrated from the off-chip memory to stacked DRAM, the stacked DRAM capacity becomes full. With no free space available to accommodate the misplaced pages in stacked DRAM, no more pages are migrated and hence the hit rate drops over time finally ending at 30.7%. The lower stacked DRAM hit rates in AutoNUMA is due to:

- The single-socket heterogeneous systems contain a non-proportional stacked DRAM capacity unlike the multi-socket systems which contain local memory capacity similar to that of remote memory. Hence, there is a high chance that AutoNUMA can find free memory available in the case of multi-socket system compared to a single-socket stacked DRAM system. As a result, AutoNUMA optimized for multi-socket systems does not consider evicting pages from local memory to accommodate misplaced pages to be migrated from the remote memory.
- Even if there is no free space available in the multi-socket system to migrate pages, AutoNUMA increases the local memory accesses by migrating the task to remote socket. However, this is not feasible in single-socket systems.
- Even in a system with higher free memory space available, since the numa_balancing_scan_period is on the order of milliseconds, the page migration happens at a very coarse – millions of CPU cycles – granularity, as identified in [25].

B. Memory Free Space over Time

Figure 3 shows the free memory space over time in a system with 24GB overall DRAM capacity. These experiments are conducted on an Intel Xeon CPU E5-2620 (more details about the machine configuration can be found in [39]) running workloads sequentially one after the other spanning over more than 2 days. Our system employs a "Samsung 850 pro" 256GB SSD as the secondary storage and hence the page-faults are serviced by a low-latency SSD. Each workload in this experiment contains 12 copies of the same application executed in the rate mode [40]. The applications chosen are from SPEC2006 [41], NAS [42], stream [43], and Mantevo [44] suites. The applications used in the workloads for Figure 3 can be observed on the X-axis in Figure 4. The memory free information is collected using the numastat [45] tool in Linux, periodically once every 2 mins. Figure 3 shows that workloads exhibit varying demands on memory over time. The free space varies from a few MegaBytes(MBs) to several GigaBytes(GBs). As can be observed, memory allocation/de-allocation varies rapidly in the order of few minutes during some phases of the experiment while not so infrequently during the other phases of the experiment thereby capturing different execution behaviour of the workloads.

C. Impact of Memory Capacity on Performance

As Figure 3 shows, different workloads are affected differently based on the OS-visible memory capacity. Figure 4 shows the ramifications of limiting the overall capacity of a workload as the overall OS-visible capacity is varied from 16GBs to 28GBs, in steps of 2GB, on an Intel Xeon CPU machine [39]. Some applications are agnostic to this variation as their entire memory footprint fits into smaller memory capacity. However, some workloads are sensitive to the overall capacity. The performance improvements (denoted by %Imp) for each workload reported in Figure 4 are *normalized* to a system with 16GB capacity and is calculated as follows:

$$\%Imp_{xGB} = \frac{((G.Mean\ Exec.Time)_{16GB} - (G.Mean\ Exec.Time)_{xGB}) * 100}{(G.Mean\ Exec.Time)_{16GB}}. \quad (1)$$

As the capacity increases from 18GB to 24GB, the average execution time improvement across all the workloads improves from 29.5% to 75.4%, saturating at 75.4%[4] for 26GB and 28GB capacities. The results in Figure 5 help us understand the variation of performance with the memory capacity. The results on the primary and secondary Y-axes represent the average number of page-faults (in millions) encountered by the workload and the average CPU utilization of the tasks, respectively, at the corresponding memory capacity. With the increase in capacity, the average number of page-faults decreases and the average CPU utilization increases. At lower capacities, most of the time is spent swapping the pages between the DRAM and secondary storage, resulting in poor CPU utilization as the tasks wait in Uninterruptible ("D") state in Linux during the swap. As the capacity increases, page-faults reduce, thereby enabling the tasks to continue in Running ("R") state resulting in 100% CPU utilization. From Figures 4 and 5, we can conclude that insufficient memory capacity can result in severe performance degradation. It is also clear from Figure 3 that memory demand in a system is workload-dependent.

D. Stacked DRAM as A Cache

While a cache adapts quickly to changing workload behavior, dedicating a memory region as a cache can significantly degrade performance. For example, in Figure 3, using 6GB out of the total 24GB capacity as a cache can cause severe performance degradation for workloads operating in regions ❶, ❷, ❸, ❹ and ❺. This is because the overall OS-visible capacity of the system will only be 18GB causing the workloads with higher

[4]As depicted in Equation 1, we used the Geometric Mean of execution times of all applications in a workload to calculate the performance improvement.

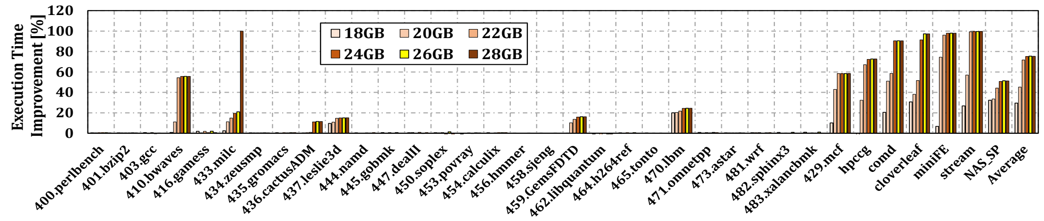


Fig. 4: Impact of capacity on overall system performance, normalized to a system with 16GB overall capacity.

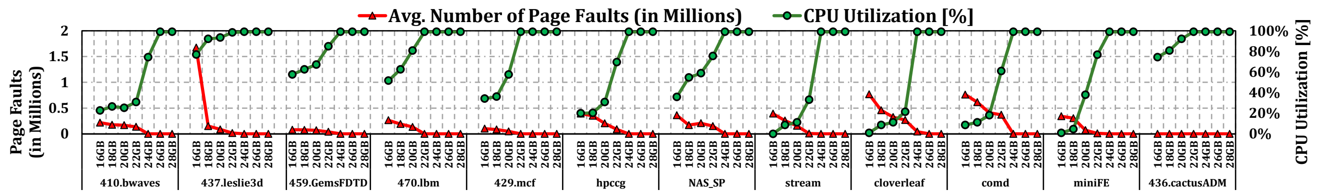


Fig. 5: Impact of capacity on page-faults and CPU Utilization for high memory footprint workloads.

memory footprints to experience page-faults. A 4GB stacked DRAM cache limits the overall capacity to 20GB, degrading workloads operating in regions marked by ❶, ❷, ❸ and ❺. Even a 2GB stacked DRAM cache would degrade workloads in regions ❶ and ❺. Hence, depending on the capacity of stacked DRAM, the decision of using stacked DRAM as a cache can have adverse-effects on performance of some workloads.

E. Stacked DRAM as PoM

To avoid the memory capacity loss incurred by caches, stacked DRAM can be used as Part of OS-visible Memory (PoM). As covered in Section I, PoM architectures: (1) enable higher throughput in datacenters as the datacenter schedulers can schedule more jobs due to increased overall memory capacity, (2) reduce the number of page-faults enabling CPUs to remain in "Running" state as covered in Section III-C, and (3) reduce the overall memory cost and power.

Though PoM architectures are beneficial, the current PoM proposals are not optimal as they substantially increase the demand for both on-chip and off-chip memory bandwidth. For example, for a stacked DRAM with capacity 6GB out of 24GB, workloads operating in all the regions except ❶ - ❺ will have to swap segments between stacked and off-chip DRAM on every stacked DRAM miss. Since PoM proposals are not cognizant of the unallocated OS addresses that are part of the OS free list containing invalid data, they still move the invalid data in these unallocated addresses during a swap operation thereby wasting the memory bandwidth. For smaller (e.g., 2GB, 4GB) stacked DRAMs, swaps and wasted bandwidth would increase further. Hence, the static decision of designing a stacked DRAM as PoM can result in free-space agnostic swapping, resulting in severe performance degradation, as will be demonstrated in Section VI.

F. Ideal Heterogeneous Memory System

An ideal heterogeneous memory design would *dynamically* provide maximum performance by:

- Reducing page-faults for capacity-limited workloads by dynamically operating in the part-of-memory (PoM) mode.
- Optimizing the overhead of swaps for the OS-visible free space by operating in the cache mode.
- Optimizing the meta-data (remapping table) overheads.

We propose a novel hardware-software co-design based system which dynamically reconfigures the heterogeneous memory system based on the overall system state. Our proposed system opportunistically converts the OS-visible free space in the system to be used as a hardware-managed cache, while switching to the part-of-memory mode for capacity-limited workloads. Based on the free space available, our co-design can operate certain memory regions in PoM mode while operating the rest in cache mode. We propose two incarnations of our co-design: (1) CHAMELEON and (2) CHAMELEON-Opt.

Algorithm 1: OS Memory Allocator Routine

```
struct page * __alloc_pages(gfp gfp_mask, unsigned int order, struct
  zonelist zonelist, nodemask_t nodemask) {
...
page = get_page_from_freelist(gfp_mask, order);
...
if (page == NULL) {
    page = alloc_pages_slowpath(gfp_mask, order);
}
...
if (page != NULL) {
    pageNum = page_to_pfn(page);
    pageSize = 0;
    if (gfp_mask contains (GFP_TRANSHUGE or
      GFP_TRANSHUGE_LIGHT) set) {
        pageSize = HPAGE_PMD_SIZE;
    } else {
        pageSize = PAGE_SIZE;
    }
    numIterations = pageSize/SegmentSize;
    for (i: 0, numIterations-1) {
        segmentNum = pageNum + (i * segmentSize);
        ISA_Alloc(segmentNum);
    }
}
return page;
}
static inline void ISA_Alloc(volatile void *p) {
    asm volatile ("isaalloc %0" : "+m"(*(volatile unsigned int *)p));
}
```

IV. CHAMELEON: Software Support

To communicate the allocated/unallocated[5] physical addresses to hardware, we propose two new processor ISA instructions: *ISA-Alloc* and *ISA-Free*. These instructions are used by the OS.

Apart from traditional 4KB pages [46], the OS employs transparent huge-pages (THPs) [47] and giant pages [48], to reduce the virtual memory overheads for workloads with huge

[5]We use the terms unallocation and reclamation synonymously in this paper.

memory footprint. Hence, in Chameleon, depending on the granularity of a segment in a segment-group and depending on the granularity of the page being allocated/unallocated, each call to the OS memory allocator/reclamation (free) routines can result in multiple corresponding ISA-Alloc/ISA-Free invocations. In a Chameleon system, Algorithms 1 and 2 present the OS memory allocator and free routines instrumented with ISA-Alloc and ISA-Free invocations. In line 12 of Algorithm 1, the "gfp_mask" flag in Linux contains the necessary bits to identify the corresponding granularity of allocation. GFP_TRANSHUGE and GFP_TRANSHUGE_LIGHT flags represent the THP allocation requests in Linux and help us detect the granularity of allocation. The ISA-Free invocation can be observed in line 17 of Algorithm 2.

Algorithm 2: OS RECLAMATION ROUTINE

```
1  static inline void __free_one_page(struct page * page, unsigned long
     pfn, struct zone * zone, unsigned int order, int migratetype) {
2  pageNum = page_to_pfn(page);
3  ...
4  list_add(&page->lru,
     &zone->free_area[order].free_list[migratetype]);
5  zone->free_area[order].nr_free++;
6  if(page != NULL) {
7      pageNum = page_to_pfn(page);
8      pageSize = 0;
9      if (order == HPAGE_PMD_ORDER) {
10         pageSize = HPAGE_PMD_SIZE;
11     } else {
12         pageSize = PAGE_SIZE;
13     }
14     numIterations = pageSize/segmentSize;
15     for(i: 0, numIterations-1) {
16         segmentNum = pageNum + (i * segmentSize);
17         ISA_Free(segmentNum);
18     }
19 }
20 return 0;
21 }
22 static inline void ISA_Free(volatile void *p) {
23     asm volatile ("isafree %0" : "+m"(*(volatile unsigned int *)p));
24 }
```

The "segmentSize" in Algorithms 1 and 2 refers to the segment size employed by Chameleon. The various segment granularities supported by the hardware in Chameleon can be easily detected by the OS during boot time. Based on the segmentSize and the allocation granularities, the number of ISA-Alloc and ISA-Free invocations vary. For a 2MB THP allocation/reclamation and for a 2KB segment employed in PoM [25], ISA-Alloc/ISA-Free is invoked 1024 times. However, for a 64-byte cache-line segment employed by CAMEO [22], ISA-Alloc/ISA-Free needs to be invoked 32,768 times.

V. CHAMELEON: HARDWARE SUPPORT

Chameleon relies on ISA-Alloc and ISA-Free calls from the OS to dynamically reconfigure regions in heterogeneous memory between Part-of-Memory (PoM) and cache modes.

Before presenting Chameleon's hardware changes, it is useful to explain the hardware design of the existing PoM proposals. The baseline PoM proposals [22, 25] employed a meta-data hardware structure referred to as Segment Remapping Table (SRT) in [25] and Line Location Table (LLT) in [22] to remap the segments between stacked and off-chip DRAMs. A set of segments in stacked and off-chip DRAMs are grouped to form a "Segment Group" in [25] ("Congruence Group" in [22]). The tag-bits in the SRT (or LLT) are unique to

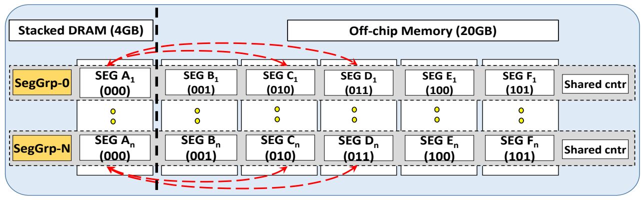


Fig. 6: Segment Restricted Remapping in PoM architectures.

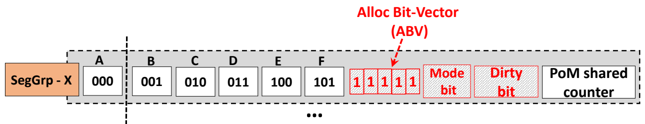


Fig. 7: Segment Restricted Remapping Table (SRRT) Entry used in our Chameleon co-design.

a segment in a segment group and aid in detecting if a segment is mapped to stacked or off-chip DRAMs at any given instant. To reduce the meta-data overheads involved in tracking the remapped segments, a segment in the stacked DRAM is only allowed to swap with another segment in the off-chip memory from the same segment group depicted by the red lines in Figure 6. Hence, the PoM proposals employ Segment Restricted Remapping Tables (SRRT) to track the hardware remapped segments. While the PoM proposal in [22] employed 64-byte cacheline segments, [25] employed 2KB segments to be swapped between the stacked and off-chip DRAMs. The SRRT in [25] uses a "Shared counter" which aids in swapping the most frequently used off-chip segment with the corresponding stacked DRAM segment. A larger 2KB segments in [25] reduces the amount of meta-data required by the SRRT and improves the spatial locality, thereby allowing higher stacked DRAM hit rates compared to [22]. Hence, our Chameleon uses the PoM architecture in [25] as a "baseline".

In Chameleon, we augment the SRRT in [25] as shown in Figure 7 with additional hardware information for each segment group. The augmented data structures are shown in red color (pattern) in Figure 7. Apart from the remapping tag bits and the shared counter in [25], each SRRT entry contains an Alloc Bit Vector (ABV), a mode bit, and a dirty bit.

Alloc Bit Vector (ABV). For each segment group, the Alloc Bit Vector (ABV) indicates whether the corresponding segments are currently allocated by the OS or not. If a segment is allocated, the corresponding bit in the ABV is set to 1; otherwise it is set to 0 indicating the segment is free. The number of entries in the ABV is equal to the number of segments per segment group, and hence is a function of the capacity ratio between the stacked and off-chip memories. When the system is initially booted, all the bits in the ABV are set to 0, and the bits will be set to 1 when the OS calls ISA-Alloc and reset to 0 when the OS calls ISA-Free.

Mode Bit. The mode bit indicates the operating mode of the segment group. It is set to 0 if the segment group is operating in the part-of-memory (PoM) mode, and is set to 1 if it is operating in the cache mode. We discuss the transitions between PoM and cache modes later in this section.

Dirty Bit. If a segment group is in the cache mode, the dirty bit in the SRRT indicates if the segment currently residing in the stacked DRAM is dirty (1) or not (0). As a result, the dirty

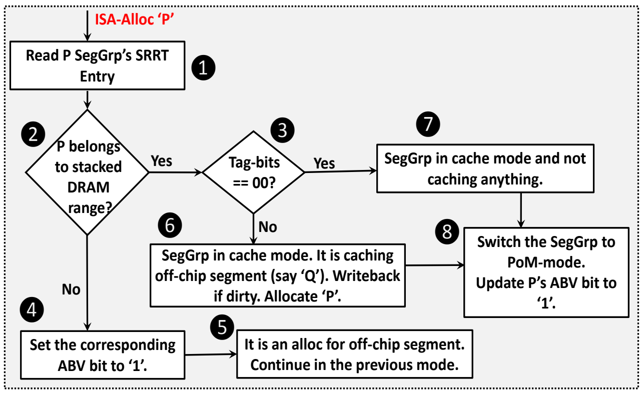

ISA-Alloc 'P'
Read P SegGrp's SRRT Entry
1
2
P belongs to stacked DRAM range?
Yes
3
Tag-bits == 00?
Yes
7
SegGrp in cache mode and not caching anything.
No
6
SegGrp in cache mode. It is caching off-chip segment (say 'Q'). Writeback if dirty. Allocate 'P'.
8
Switch the SegGrp to PoM-mode. Update P's ABV bit to '1'.
No
4
Set the corresponding ABV bit to '1'.
5
It is an alloc for off-chip segment. Continue in the previous mode.

Fig. 8: Chameleon ISA-Alloc Transition Flowchart.

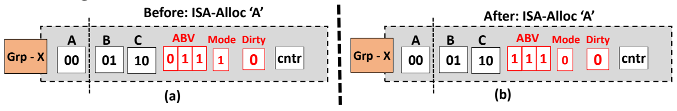


Fig. 9: Chameleon ISA-Alloc Transition (Example).

bit indicates if the segment needs to be written back to the off-chip memory during eviction from the stacked DRAM. If the segment group is operating in the PoM mode, the dirty bit is ignored, indicating that the segments will be swapped irrespective of the status of the dirty bit.

A. Chameleon Cache and PoM Modes

At a high level, an ISA-Alloc instruction can transition segment group(s) from the cache mode to PoM mode, while an ISA-Free instruction can transition the segment group from the PoM mode to the cache mode. However, not all the ISA-Alloc and ISA-Free instructions for the physical addresses in a segment group will trigger these transitions. The number of currently allocated/unallocated segments in a segment group governs whether an ISA-Alloc or ISA-Free instruction will trigger a transition or not.

Before describing the scenarios which trigger transitions between PoM and cache modes, we explain the notation in the SRRT shown in Figure 7. Segments A, B, C, D, E and F shown in the SRRT represent the actual physical segments that are part of a segment group represented by SegGrp-X in the figure. In a system with 4GB of stacked DRAM and 20GB of off-chip DRAM, a ratio of 1:5 implies we need six segments in a segment group. A segment in stacked DRAM will have a physical address in the range [0, 0xFFFFFFFF], while off-chip segments will belong to the range [0x100000000, 0x5FFFFFFFF]. The tag bits in Figure 7 signify where a corresponding physical address is remapped to (or cached at) in a segment group. We next describe two variations of our proposed architecture with help of flowcharts and relevant examples with 1:2 capacity ratio (with a total of 3 segments in segment group) for simplicity.

B. Chameleon Design

In our basic Chameleon architecture, we leverage the OS-visible free space in the stacked DRAM as a cache, but not the free space in the off-chip DRAM. As a result, transitions from the PoM to cache mode and vice versa are only triggered

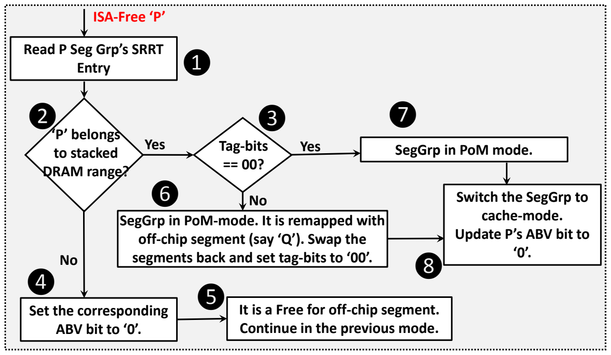

ISA-Free 'P'
Read P Seg Grp's SRRT Entry
1
2
'P' belongs to stacked DRAM range?
Yes
3
Tag-bits == 00?
Yes
7
SegGrp in PoM mode.
No
6
SegGrp in PoM-mode. It is remapped with off-chip segment (say 'Q'). Swap the segments back and set tag-bits to '00'.
Switch the SegGrp to cache-mode. Update P's ABV bit to '0'.
8
No
4
Set the corresponding ABV bit to '0'.
5
It is a Free for off-chip segment. Continue in the previous mode.

Fig. 10: Chameleon ISA-Free Transition Flowchart.

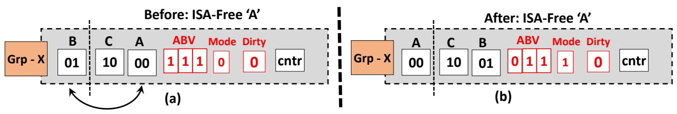


Fig. 11: Chameleon ISA-Free Transition (Example).

by ISA-Alloc/ISA-Free for physical addresses belonging to the stacked DRAM address range.

1) ISA-Alloc Transitions: Figure 8 shows the flowchart for ISA-Alloc transitions in Chameleon. If the ISA-Alloc is for an off-chip DRAM physical address like the flow ❶ → ❷ → ❹ → ❺ in Figure 8, the segment group continues to operate in the previous mode. However, if the ISA-Alloc is for a stacked DRAM physical address, the segment group will be operating in the cache mode, and the segment in stacked DRAM could be caching an off-chip segment or not. Figure 9(a) represents a scenario where ISA-Alloc is encountered when none of the off-chip segments are cached in the stacked DRAM (represented by tag bits: 00). In such a scenario, ISA-Alloc will follow the flow: ❶ → ❷ → ❸ → ❼ → ❽. From Figure 9(a), since segment A is not allocated, the ABV for A is still 0 while the segment group (Grp-X) is operating in cache mode (mode bit: 1). After ISA-Alloc, the ABV for A is set to '1' and the segment group transitions to PoM mode (Figure 9(b)).

For the scenario where an off-chip segment is cached in the stacked DRAM indicated by the tag bits: If the dirty bit is set, the corresponding segment is written back to the original segment, otherwise the tag bits can be simply over-written. This is represented by the flow: ❶ → ❷ → ❸ → ❻ → ❽ in Figure 8. Finally, the ABV for the stacked DRAM segment is set to '1' indicating that it is allocated.

2) ISA-Free Transitions: Figure 10 shows the flowchart for ISA-Free transitions in our Chameleon design. For an ISA-Free to off-chip physical address, there is no transition in segment group modes, but the corresponding ABV bit is set to '0'.

If an ISA-Free is for a stacked DRAM physical address, the segment group will be operating in the PoM mode prior to the call. There are two scenarios for ISA-Free depending on whether the segment to be freed is remapped or not. If the tag bits indicate the segment to be freed is not remapped, the corresponding ABV bit is set to '0' and the segment group transitions to cache mode following the flow: ❶ → ❷ → ❸ → ❼ → ❽. The tag bits are set to '00'; indicating that no segments are cached in the stacked DRAM.

If the segment to be freed is currently not in stacked

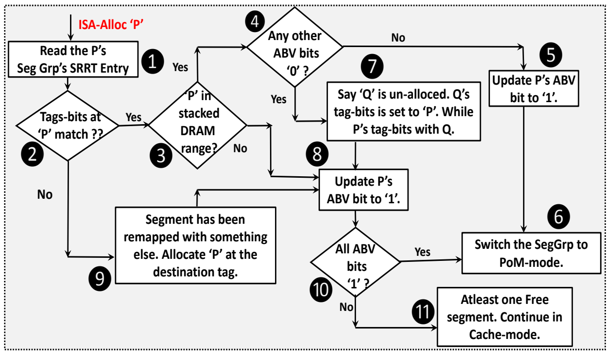

ISA-Alloc 'P'
Read the P's Seg Grp's SRRT Entry
1
Tags-bits at 'P' match ??
2
Yes
No
'P' in stacked DRAM range?
3
Yes
No
Any other ABV bits '0' ?
4
No
Yes
Update P's ABV bit to '1'.
5
Switch the SegGrp to PoM-mode.
6
Say 'Q' is un-alloced. Q's tag-bits is set to 'P'. While P's tag-bits with Q.
7
Update P's ABV bit to '1'.
8
Segment has been remapped with something else. Allocate 'P' at the destination tag.
9
All ABV bits '1' ?
10
Yes
No
Atleast one Free segment. Continue in Cache-mode.
11

Fig. 12: Chameleon-Opt ISA-Alloc Transition Flowchart.

DRAM (Figure 11(a)), the original stacked DRAM segment 'A' is remapped to off-chip DRAM segment 'C', while 'C' is remapped to 'B'. This can happen with the following re-mappings. Initially the state is: A, B and C, with A in stacked DRAM, and B and C in off-chip DRAM. If segment C is accessed more frequently, based on the fast swaps implemented in [25], C will be swapped with A transitioning to state: C, B and A. This ensures that the most frequently accessed segment C resides in stacked DRAM. If segment B is accessed more frequently, in the next program phase, segments C and B would be swapped leading to the state in Figure 11(a) . Now, if ISA-Free happens for segment A which is the original stacked DRAM segment, segment A needs to be proactively swapped with the current segment B in stacked DRAM before it is freed to ensure the stacked DRAM segment is available for caching. Finally, after swapping, as shown in Figure 11(b), segment A's ABV is set to '0' and the segment group transitions to cache mode from PoM mode following the flow: ❶ → ❷ → ❸ → ❻ → ❽.

C. *Optimized Chameleon (Chameleon-Opt) Design*

The basic Chameleon design can only leverage OS-visible free space in stacked DRAM to be used as a cache even though there are available free segments in off-chip memory. Consequently, Chameleon does not optimally leverage all available free space in the system. To increase cache capacity, we present an optimized co-design, Chameleon-Opt, which can proactively remap segments in the stacked DRAM to the off-chip memory. Such a design can convert the free space available in both the stacked and off-chip DRAMs for caching. This Chameleon-Opt design outperforms the Chameleon design, as will be demonstrated in Section VI.

1) ISA-Alloc Transitions: Figure 12 shows the flowchart for an ISA-Alloc instruction in Chameleon-Opt. The ABV bits play a crucial role in Chameleon-Opt as they signify when to switch the segment group from one mode to another. At a high level, a segment group remains in cache mode as long as one of its ABV bits is 0, and it switches to the PoM mode when all the ABV bits are 1. The test condition in ❿ in Figure 12 represents this check. Hence, unlike Chameleon, a segment group in Chameleon-Opt continues in the cache mode even if its corresponding stacked DRAM address has been allocated by the OS. Similarly, it switches to the cache mode even if an off-chip physical address has been unallocated while the stacked DRAM segment still remains allocated. This is

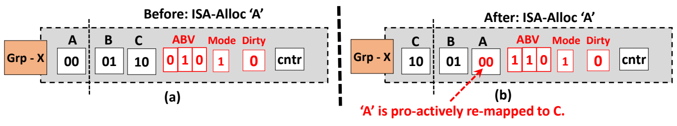


Fig. 13: Chameleon-Opt ISA-Alloc Transition Example.

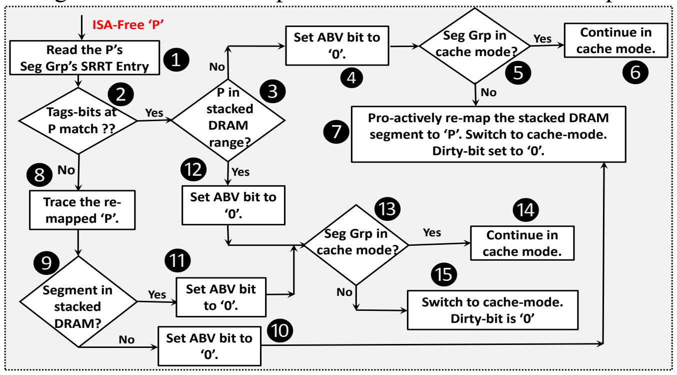


Fig. 14: Chameleon-Opt ISA-Free Transition Flowchart.

possible because Chameleon-Opt proactively remaps the current segment in stacked DRAM to free up space for caching.

In Chameleon-Opt, unlike Chameleon, the segment group which encounters an ISA-Alloc would always be operating in cache mode since encountering ISA-Alloc itself indicates the presence of a free segment in that group. The actions triggered for ISA-Alloc depend on the segment currently residing in the stacked DRAM as well as whether the ISA-Alloc is for a stacked DRAM physical address or not. Following flow: ❶ → ❷ → ❸ → ❹ → ❺ → ❻ in Figure 12, as the segment group operates in cache mode and there is no segment cached in stacked DRAM (as the tag bits match the segment being allocated). Therefore, the ISA-Alloc will directly set the ISA-Alloc'ed segments' ABV bits to '1' and transitions the segment group to PoM mode. This is because condition check at ❹ confirmed that there are no other OS-visible free segments in the segment group. Figure 13(a) shows one possible scenario for a state prior to executing ISA-Alloc instruction following flow: ❶ → ❷ → ❸ → ❹ → ❼ → ❽ → ❿ → ❻ of the flowchart in Figure 12. In this scenario, segments A and C are not allocated while B is allocated, represented by ABV bits: 0 1 0. The segment group is initially operating in cache mode (mode bit: 1). As the stacked DRAM tag bits match segment A, no other segment is currently cached in stacked DRAM. With Chameleon, for ISA-Alloc to segment A, segment A is allocated in stacked DRAM (setting its ABV bit to 1) and the segment group transitions to PoM mode. However, in Chameleon-Opt (Figure 13(b)), segment A is proactively remapped to Segment C in the off-chip DRAM. Hence, C's tag bits are set to "00" while the tag of the stacked DRAM segment is set to "10" (i.e., segment C). Since segment C is never allocated, it will never result in a stacked DRAM hit allowing for either B or A to be cached in future. As a result, in Chameleon-Opt, the segment group still operates in the cache mode unlike Chameleon.

The other flows in Figure 12 are straight-forward. For example, flow: ❶ → ❷ → ❾ → ❽ → ❿ → ❻ from Figure 12 would follow proactive remapping explained in Figure 13(b) a segment is remapped to off-chip memory, while stacked DRAM caches an off-chip segment. Since the segment group is in cache mode, an ISA-Alloc'ed segment will be allocated in stacked DRAM and the segment group transitions to PoM Mode. While in flow ❶ → ❷ → ❾ → ❽ → ❿ → ⓫, since there are more unallocated segments, the segment group continues to operate in cache mode.

Flow: ❶ → ❷ → ❸ → ❽ → ❿ → ❻ occurs when ISA-Alloc is for an off-chip address that is not remapped . In this case, the segment is allocated at the original off-chip address and the segment group transitions to the PoM mode. While in flow ❶ → ❷ → ❸ → ❽ → ❿ → ⓫, since there are more unallocated segments, the segment group continues to operate in the cache mode.

2) ISA-Free Transitions: Figure 14 shows ISA-Free transitions for Chameleon-Opt. Apart from the free space available in stacked DRAM, Chameleon-Opt proactively converts the free space available in off-chip DRAM to be visible as free space in stacked DRAM to be leveraged as a cache. The proactive free space creation in Chameleon-Opt can be observed in the flow: ❶ → ❷ → ❸ → ❹ → ❺ → ❼.

The other flow, ❶ → ❷ → ❸ → ❹ → ❺ → ❻, represents a similar scenario, except that the segment group is already in the cache mode as there are more free segments in the segment group. As a result, the segment group continues to operate in cache mode, and the corresponding ABV bit is set to '0'. The flows ❶ → ❷ → ❸ → ⓬ → ⓭ → ⓮ and ❶ → ❷ → ❸ → ⓬ → ⓭ → ⓯ represent scenarios where the ISA-Free is encountered for stacked DRAM addresses that are neither cached nor remapped with any off-chip segments, respectively. ISA-Free transitions the segment group to cache mode if it was previously operating in PoM mode. The other flows in Figure 14 correspond to cases where a segment being freed is remapped with other segments or not.

D. Additional Discussion

1) In-transit Segment Accesses: In Chameleon and Chameleon-Opt, we use local buffers proposed by PoM's fast-swap design [25] to hold segments being moved between stacked and off-chip DRAMs without OS involvement. If segments being swapped or moved are accessed, loads and stores to these in-transit segments are performed at these local buffers in the respective memories and hence do not incur longer latencies.

2) Security Concerns: In Chameleon and Chameleon-Opt, a potential security concern is information leakage when data in segments that were used as a cache are accessed by a different process when they move to PoM mode. Alternatively, segments in the PoM mode could be read by a cache-accessing spy process when they transition to cache mode. To avoid these issues, both ISA-Free and ISA-Alloc instructions trigger hardware to clear the segments that transition between cache and PoM use.

3) Impact on Buffer Cache Space: Linux manages a part of memory as buffer for secondary storage devices (e.g., hard disk or SSD). As depicted in the examples in [49], since the amount of space allocated to buffer cache impacts the total free space visible to the OS, the ISA-Alloc and ISA-Free coming from pages allocated or unallocated by the OS for this buffer cache is honored by our Chameleon hardware similar to any other allocation/de-allocation requests. As a result, our Chameleon and Chameleon-opt co-designs do not take away buffer cache space to use it as a hardware-managed cache. As a result, our Chameleon and Chameleon-Opt co-designs do not impact buffer cache space and hence do not degrade disk performance.

Cores	12 @ 3.6GHz (each), ALPHA ISA, out-of-order
L1(I/D)	32KB, 4-way associative, 64B cacheline
L2 Cache	256KB (private), 8-way associative, 64B cacheline
L3 Cache	12MB (shared), 16-way associative, MESI, 64B cacheline
Stacked DRAM (4GB)	Bus Frequency: 1.6GHz (DDR 3.2GHz), Bus Width: 128 bits/channel, Capacity: 4GB, 2 channels, 2 ranks/channel, 8 banks/rank tCAS-tRCD-tRP-tRAS: 11-11-11-28, tRFC: 138 nsecs
Off-chip DRAM (20GB)	Bus Frequency: 800MHz (DDR 1.6GHz), Bus Width: 64 bits/channel, Capacity: 20GB, 2 channels, 2 ranks/channel, 8 banks/rank tCAS-tRCD-tRP-tRAS: 11-11-11-28, tRFC: 530 nsecs
OS	Linux Kernel 4.2.2
Page-Fault Latency	100K CPU cycles (36 micro-seconds) [22] (SSD)

TABLE I: Simulated Baseline Configuration.

4) Pipeline execution details for ISA-Alloc/ISA-Free: The ISA-Alloc/ISA-Free instructions retire based on the transitions triggered by these instructions. If the ISA-Alloc/ISA-Free instructions do not necessitate any segment movements, these instructions are retired as soon as the ABV-bits in the SRRT are updated. Clearing out segments could occur after ISA-Alloc/ISA-Free retire but before they are accessed in a different mode. However, if the ISA-Alloc/ISA-Free involve moving segments, then these instructions are committed as soon as the segments being moved are fetched into the local buffers of the corresponding memory controllers to ensure they can be accessed in-flight. The segments in the local buffers are copied to the destination memory controllers write buffer which are drained opportunistically. In our evaluation, such a design caused an overhead of 1.06% on average (Section VI-F).

VI. Evaluation

A. Experimental Setup

As discussed briefly in Section III, we used GEM5 full-system simulator [36] executing Linux kernel-4.2.2 with stacked and off-chip DRAMs modeled using the memory controller support in GEM5. ISA-Alloc/ISA-Free are invoked from the OS memory allocator/reclamation code using the pseudo-instruction support in GEM5 [50]. Table I summarizes our simulated configuration. We simulated applications from various suites discussed in Section III-B whose characteristics are presented in Table II. Our workloads are fast-forwarded to the region of interest and caches are warmed-up. We simulate 500 million instructions per application and with 12 applications in a workload, we simulated a minimum of 6 billion (500M*12) total instructions . We assume a page fault latency of 10^5 CPU cycles (serviced by SSDs). All the reported results are for 4GB stacked and 20GB off-chip DRAM, unless otherwise stated. In all the results reported in Section VI-B, the performance reported is the *geometric mean* of Instructions committed Per Cycle (IPC) of all the benchmarks in a workload *normalized* to the corresponding baselines.

Suite	WL	LLC-MPKI	MF (GBs)	Suite	WL	LLC-MPKI	MF (GBs)
SPEC2006	bwaves	12.91	21.86	Mantevo	cloverleaf	30.33	23.01
	lbm	29.55	19.17		comd	0.71	23.18
	cactusADM	2.03	20.12		miniAMR	1.44	22.40
	leslie3d	12.18	21.65		hpccg	7.81	22.15
	mcf	59.804	19.65		miniFE	0.48	22.55
	GemsFDTD	20.783	22.56		miniGhost	0.19	20.68
NAS	SP	0.87	21.72	Stream	Stream	35.77	21.66

TABLE II: Workload Characteristics.(MF: Memory Footprint, WL: Workload, MPKI: Misses Per Kilo Instructions)

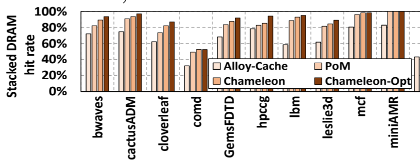


Fig. 15: Stacked DRAM hit rate results.

B. Results

Figure 16 shows the breakdown of the percentage of segment groups operating in cache mode and PoM mode in Chameleon and Chameleon-Opt designs. This distribution should ideally vary over time with allocation/unallocation requests in the workload. However, in our workloads, the applications allocate at the beginning and unallocate at the end of their execution, as a result, we did not encounter ISA-Alloc and ISA-Free in our simulated snippets. On average, 9.2% of the segment groups operate in the cache mode in Chameleon compared to 40.6% in Chameleon-Opt. This is because Chameleon-Opt leverages the free space available in the off-chip DRAM to be used as a cache. Figure 15 presents the stacked DRAM hit rate for the latency-optimized Alloy Cache [14], PoM and both Chameleon designs. Since Alloy Cache employs a latency-optimized direct-mapped cache design with 64B lines, it has the lowest stacked DRAM average hit rate of 62.4%, while PoM with 2KB segments has an average hit rate of 81%. In comparison, Chameleon and Chameleon-Opt have average hit rates of 84.6% of 89.4%, respectively. This higher hit rates in Chameleon and Chameleon-Opt over PoM is due to more segment groups operating in cache mode. As discussed in Section III-E, PoM employs a "threshold" which signifies the minimum number of accesses to an off-chip DRAM segment before it can be swapped with a stacked DRAM segment. Since Chameleon does not employ any such threshold for segment groups operating in the cache mode, it has a higher stacked DRAM hit rate compared to PoM. Since more such segment groups operate in cache mode in Chameleon-Opt, its hit rate is higher than Chameleon. Figure 17 quantifies the number of swaps^2 incurred in PoM, Chameleon and Chameleon-Opt designs. The results reported are *normalized* to the number of swaps incurred in PoM. Due to many segment groups operating in the cache mode, the overall number of swaps is reduced on an average by 14.4% and 43.1% in Chameleon and Chameleon-Opt, respectively vs. PoM. Note that for a segment group in cache mode, evicting a modified (represented by dirty bit)

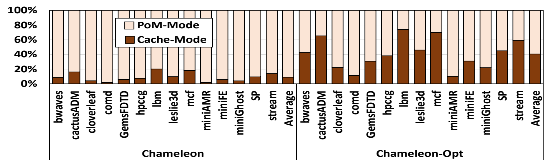


Fig. 16: PoM to Cache mode segment group distribution.

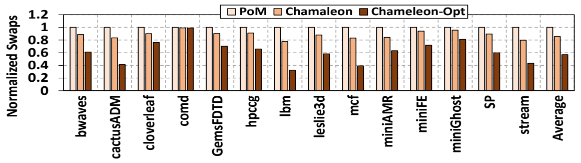


Fig. 17: Segment swaps between stacked and off-chip DRAMs.

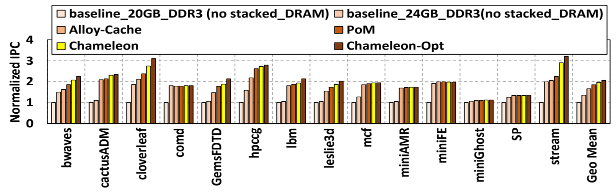


Fig. 18: Normalized IPC results.

stacked DRAM segment results in a writeback to off-chip DRAM before the stacked DRAM segment is filled with an off-chip segment. This is effectively still a swap, as the writeback of the modified segment still consumes both the stacked and off-chip memories' bandwidth. Hence in our Chameleon results reported, these scenarios are still considered as swaps.

Figure 18 shows the normalized IPC of various designs including Alloy Cache, PoM, Chameleon, and Chameleon-Opt. There are two variant baseline systems, both without stacked DRAM and the total capacity coming from off-chip DRAM. While one baseline offers 20GB overall capacity, the other offers 24GB overall capacity. The 24GB baseline does not incur any page faults unlike the 20GB capacity. The 24GB baseline system improves the Geometric Mean of IPC by 35.6% over the 20GB capacity baseline. As mentioned in Section VI-A, Alloy Cache, PoM, Chameleon and Chameleon-Opt configurations all use a 4GB stacked and 20GB off-chip DRAM, thereby having a total capacity of 24GB. Alloy Cache experiences page-faults for workloads with high memory footprints similar to both baselines since it sacrifices total capacity to use as a cache. As a result, though it improves the performance over the baselines, its performance is lower than other alternatives for many workloads, as shown in Figure 18. PoM improves the performance (Geometric Mean of IPC) over the 20GB and 24GB capacity baselines by 85.2% and 36.5%, respectively. Chameleon improves the Geometric Mean of IPC for all the workloads by 96.8% and 45.1% over the 20GB and 24GB baseline systems respectively, and by 6.3% and 18.5% over PoM and Alloy Cache, respectively. Chameleon-Opt improves the performance by 106.3% and 52.0% over the 20GB and 24GB baseline systems respectively, and by 11.6% and 24.2% over PoM and Alloy Cache, respectively. Such an improvement over the baseline systems is mainly because Chameleon manages to cater for the high memory footprint by averting page-faults and could utilize the high bandwidth stacked DRAM more efficiently, averting slow off-chip DRAM accesses. The success of Chameleon and Chameleon-Opt over

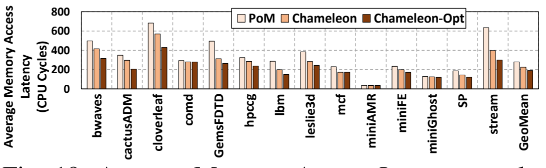


Fig. 19: Average Memory Access Latency results.

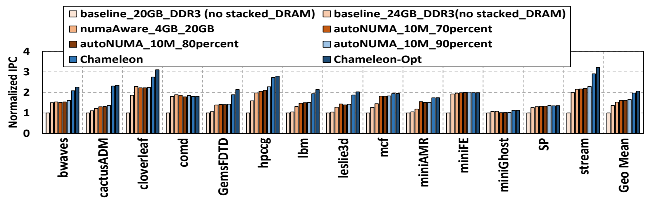


Fig. 20: Normalized IPC results comparison.

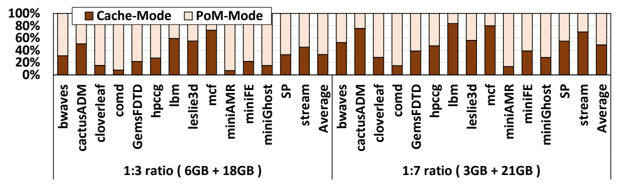


Fig. 21: Sensitivity distribution results for Chameleon-Opt.

PoM is due to the increased stacked DRAM hit rate as well as the reduced swaps between the stacked and off-chip DRAMs, which reduces the average memory access latency as shown in Figure 19. Further, Chameleon-Opt incurs minimal swaps and hence lower average memory latency as it can create more free space in the stacked DRAM compared to Chameleon, thereby outperforming it by 4.8%.

C. Comparison with OS-based solutions

As discussed in Sections III-A1 and III-A2, the NUMA-Aware Memory Allocator and AutoNUMA under-utilize stacked DRAM resulting in a stacked DRAM hit rates of 18.5% and 64.4%. Therefore, these OS-based solutions do not leverage the full potential of high-bandwidth stacked DRAM. Figure 20 shows that Chameleon has an average improvement of 28.7% and 19.1% over NUMA-Aware Memory Allocator and AutoNUMA respectively, while Chameleon-Opt improves the performance by 34.8% and 24.9%. Figures 18 and 20 show that some benchmarks (e.g., miniFE, miniGhost, comd, and SP) do not benefit from Chameleon or Chameleon-Opt. This is because their memory intensity denoted by LLC-MPKI in Table II is very low. As a result, these applications do not benefit from opportunistically converting the free space in to cache. On the other hand, applications like hpccg, bwaves, stream and cloverleaf which are highly memory intensive benefit significantly from Chameleon and Chameleon-Opt.

D. Comparison with Polymorphic Memory

A patent by Chung, et al. [51] proposed a hybrid architecture which can leverage the free memory available in stacked DRAM as cache. Their proposed architecture could only leverage the OS-visible free-space in stacked DRAM and not the off-chip DRAM for caching (unlike Chameleon-Opt). Though this proposal achieves the same amount of free space as our original Chameleon design, our basic Chameleon still outperforms their proposal by 10.5% as can be observed in Figure 22. This is because the Polymorphic Memory proposal does not swap the most frequently used pages from the off-chip DRAM with the ones in stacked DRAM for OS allocated pages (unlike PoM), thereby under-utilizing the stacked DRAM. Chameleon and Chameleon-Opt improve the performance by 10.5% and 15.8% over Polymorphic Memory, respectively.

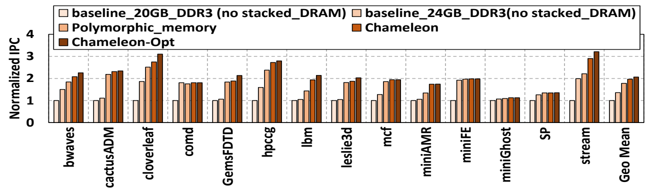


Fig. 22: Polymorphic Memory [51] Comparison.

E. Sensitivity Results

Figure 21 shows the cache and PoM mode distribution for different ratios of stacked and off-chip DRAM capacities. For a 1:3 ratio, the stacked DRAM contributes a total capacity of 6GB, while the off-chip DRAM contributes 18GB and for 1:7 ratio, stacked DRAM contributes 3GB, while the off-chip contributes 21GB. As the ratio of stacked-to-off-chip DRAM increases from 1:3 to 1:7, the average number of cache-mode segment groups increases from 33% to 48.7% vs 40.6% in 1:5 configuration. This is because as the number of segments per segment group increases from 3 to 7, the probability of finding at least one free segment increases in Chameleon-Opt, thereby increasing the cache mode segment groups. Figure 23 represents the normalized performance corresponding to these ratios, and shows that Chameleon-Opt consistently performs better across all ratios. For 1:3 ratio, Chameleon and Chameleon-Opt improve the performance by 5.9% and 7.6%, respectively, over PoM, while for 1:7 ratio, the improvements are 8.1% and 12.4%, respectively, over PoM.

F. ISA-Alloc and ISA-Free Overhead Analysis

As discussed in Sections V-B and V-C, depending on the segment-group state, ISA-Alloc and ISA-Free may initiate an additional segment swap per ISA-Alloc/ISA-Free between stacked and off-chip DRAMs due to the hardware remapping employed. As a result, the ISA-Alloc and ISA-Free can cause performance overheads due to these un-warranted swaps. To get an estimate on the overheads introduced by the ISA-Alloc and ISA-Free, we performed an overhead analysis based on conservative assumptions for end-to-end workload execution results presented in Figure 3. Assuming a 2KB segment, since Chameleon builds on the fast-swap approach proposed in [25], it encounters one swap for every ISA-Alloc and ISA-Free, thereby encountering 242.8 Million swaps over 193680 seconds (53.8 hours) of execution. Assuming the swap operation memory latency of 700 CPU cycles per 64-byte cacheline (observed for PoM in Figure 19), the total time spent by a 2.25GHz (average of 2GHz(base) and 2.5GHz(max turbo)) Intel Xeon system [39] swapping the 2KB segments between the stacked and off-chip DRAMs is: $(242.8*10^6*700*2048/64*2.25*10^9)$=2071.89 seconds. Thus, the overheads encountered by Chameleon due to ISA-Alloc and ISA-Free accounts to (2071.89*100/193680)=1.06% of the end-to-end execution time. This shows that the additional hardware transitions warranted by Chameleon do not pose significant performance overheads.

G. Limitations and Future Work

Our Chameleon and Chameleon-Opt architectures opportunistically convert the free space in the heterogeneous memory

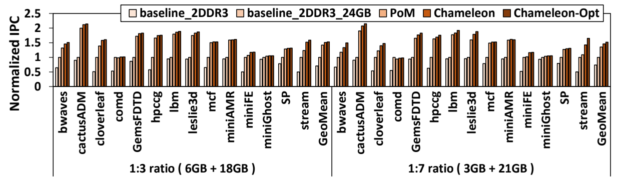


Fig. 23: Sensitivity IPC results for Chameleon-Opt.

system to be used as cache. The baseline PoM architecture [25] employs Segment Restricted Remapping as briefly covered in section V to reduce the meta-data overheads. Both Chameleon and Chameleon-Opt employ the same Segment Restricted Remapping, which limits their ability to maximize cache capacity if free space is uneven between different segment groups. For example, a segment group might have two free segments while another segment group has none and cannot operate in the cache mode. To alleviate this limitation, we could expose the segment group management information to OS so that OS can maintain additional data structures to keep track of the ABV bits per segment group in a separate data-structure updated during ISA-Alloc and ISA-Free execution. This could be explored as part of the future work. Another Chameleon/Chameloen-Opt limitation is due to the use of segment-sized blocks. Larger segments benefit workloads that take advantage of spatial locality. A different proposal, CAMEO [22] uses 64B block granularities thereby reducing the data movement and swapping overhead, which benefits workloads with limited spatial locality. However, CAMEO incurs higher remapping table overheads due to 64B block organization.

VII. Conclusion

In this paper, we propose Chameleon, a novel co-designed architecture which dynamically re-configures a heterogeneous memory system based on the available free-space. Using two new instructions (ISA-Alloc, ISA-Free), the OS informs hardware of pages that have been allocated or freed. Based on this OS communication, Chameleon adapts dynamically by operating in Part-of-Memory mode for high memory footprint workloads, while opportunistically converting the free-space to be used as a hardware-managed cache. As a result, Chameleon ensures better performance compared to the state-of-the-art part-of-memory and cache proposals.

Acknowledgment

The authors would like to thank Jaewoong Sim for the initial discussions and anonymous reviewers for their valuable feedback. This work is supported in part by NSF grants 1822923, 1439021, 1629915, 1626251, 1629129, 1763681, 1526750 and 1439057. AMD, the AMD Arrow logo, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

References

[1] "AMD HBM," https://goo.gl/g2gWL7.
[2] "Micron HMC," https://goo.gl/F3kXvm.
[3] "Intel Xeon-Phi," https://goo.gl/M9piiE.
[4] J. Macri, "AMD's next generation GPU and high bandwidth memory architecture: FURY," in *2015 IEEE Hot Chips 27 Symposium (HCS)*, 2015.
[5] J. B. Kotra, N. Shahidi, Z. A. Chishti, and M. T. Kandemir, "Hardware-software co-design to mitigate dram refresh overheads: A case for refresh-aware process scheduling," in *Proceedings of 22nd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2017.
[6] "AMD Vega," https://goo.gl/moSqgH.
[7] 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 (ISCA)*, 2015.
[8] H. Jang, Y. Lee, J. Kim, Y. Kim, J. Kim, J. Jeong, and J. W. Lee, "Efficient footprint caching for Tagless DRAM Caches," in *Proceedings of the 2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2016.
[9] D. Jevdjic, G. H. Loh, C. Kaynak, and B. Falsafi, "Unison Cache: A Scalable and Effective Die-Stacked DRAM Cache," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2014.
[10] D. Jevdjic, S. Volos, and B. Falsafi, "Die-stacked DRAM Caches for Servers: Hit Ratio, Latency, or Bandwidth? Have It All with Footprint Cache," in *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA)*, 2013.
[11] X. Jiang, N. Madan, L. Zhao, M. Upton, R. Iyer, S. Makineni, D. Newell, Y. Solihin, and R. Balasubramonian, "CHOP: Adaptive filter-based DRAM caching for CMP server platforms," in *In Proceedings of the 16th International Symposium on High-Performance Computer Architecture (HPCA)*, 2010.
[12] 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 (ISCA)*, 2015.
[13] 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 (MICRO)*, 2011.
[14] M. K. Qureshi and G. H. Loh, "Fundamental Latency Trade-off in Architecting DRAM Caches: Outperforming Impractical SRAM-Tags with a Simple and Practical Design," in *Proceedings of the 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2012.
[15] S. Shin, S. Kim, and Y. Solihin, "Dense Footprint Cache: Capacity-Efficient Die-Stacked DRAM Last Level Cache," in *Proceedings of the Second International Symposium on Memory Systems (MEMSYS)*, 2016.
[16] J. Sim, G. H. Loh, H. Kim, M. O'Connor, and M. Thot-

tethodi, "A Mostly-Clean DRAM Cache for Effective Hit Speculation and Self-Balancing Dispatch," in *Proceedings of the 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2012.

[17] J. B. Kotra, M. Arjomand, D. Guttman, M. T. Kandemir, and C. R. Das, "Re-NUCA: A practical nuca architecture for reram based last-level caches," in *Proceedings of IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2016.

[18] O. Kislal, M. T. Kandemir, and J. Kotra, "Cache-aware approximate computing for decision tree learning," in *Proceedings of IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2016.

[19] J. Liu, J. Kotra, W. Ding, and M. Kandemir, "Network footprint reduction through data access and computation placement in noc-based manycores," in *Proceedings of the 52Nd Annual Design Automation Conference (DAC)*, 2015.

[20] J. B. Kotra, D. Guttman, N. C. N., M. T. Kandemir, and C. R. Das, "Quantifying the potential benefits of on-chip near-data computing in manycore processors," in *Proceedings of 25th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2017.

[21] P. Yedlapalli, J. Kotra, E. Kultursay, M. Kandemir, C. R. Das, and A. Sivasubramaniam, "Meeting midway: Improving cmp performance with memory-side prefetching," in *Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2013.

[22] 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)*, 2014.

[23] M. Oskin and G. H. Loh, "A Software-Managed Approach to Die-Stacked DRAM," in *Proceedings of the 2015 International Conference on Parallel Architecture and Compilation (PACT)*, 2015.

[24] J. H. Ryoo, K. Ganesan, Y. M. Chen, and L. K. John, "i-MIRROR: A Software Managed Die-Stacked DRAM-Based Memory Subsystem," in *Proceedings of the 27th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, 2015.

[25] 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)*, 2014.

[26] "ESXi White paper," https://goo.gl/5tQQBk.

[27] "NUMA-aware Allocation," https://goo.gl/PrRgLQ.

[28] J. B. Kotra, S. Kim, K. Madduri, and M. T. Kandemir, "Congestion-aware memory management on numa platforms: A vmware esxi case study," in *Proceedings of IEEE International Symposium on Workload Characterization (IISWC)*, 2017.

[29] "AutoNUMA," https://lwn.net/Articles/488709/.

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

[31] 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 *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, 2015.

[32] X. Yu, C. J. Hughes, N. Satish, O. Mutlu, and S. Devadas, "Banshee: Bandwidth-efficient DRAM caching via software/hardware cooperation," *CoRR*, vol. abs/1704.02677, 2017.

[33] A. Prodromou, M. Meswani, N. Jayasena, G. Loh, and D. M. Tullsen, "MemPod: A Clustered Architecture for Efficient and Scalable Migration in Flat Address Space Multi-level Memories," in *Proceedings of the 2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2017.

[34] J. H. Ryoo, M. R. Meswani, A. Prodromou, and L. K. John, "Silc-fm: Subblocked interleaved cache-like flat memory organization," in *Proceedings of the 2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2017.

[35] "Intel MC-DRAM," https://goo.gl/Z4UiKo.

[36] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The Gem5 Simulator," *SIGARCH Comput. Archit. News*, 2011.

[37] "Linux NUMA Emulation," https://goo.gl/Zsv9CG.

[38] "numactl," https://linux.die.net/man/8/numactl.

[39] "Intel Xeon Machine," https://goo.gl/USH8dD.

[40] "Rate Mode," https://goo.gl/i4148Z.

[41] "SPECCPU 2006," https://www.spec.org/cpu2006/.

[42] "NAS Benchmark," https://goo.gl/jQvMKbl.

[43] "Stream Benchmark," https://www.cs.virginia.edu/stream/.

[44] M. A. Heroux, D. W. Doerfler, P. S. Crozier, J. M. Willenbring, H. C. Edwards, A. Williams, M. Rajan, E. R. Keiter, H. K. Thornquist, and R. W. Numrich, "Improving Performance via Mini-applications," in *Sandia National Laboratory Technical Report*, 2009.

[45] "numastat," https://goo.gl/M8U5hN.

[46] "Traditional Pages," https://goo.gl/zgCHKQ.

[47] "Transparent Huge Pages," https://goo.gl/nrqpRT.

[48] "Giant Pages," https://goo.gl/BE6FtN.

[49] "Linux Buffer Cache," https://goo.gl/qRGa3s.

[50] "GEM5 Pseudo-instructions," https://goo.gl/XDzQcw.

[51] J. Chung and N. Soundararajan, "Polymorphic stacked dram memory architecture," Patent US 13/036,839, 2012. [Online]. Available: https://www.google.ch/patents/US20120221785