NEXT-GEN HYBRID MEMORY AND INTERCONNECT SYSTEM ARCHITECTURES

A Dissertation

by

FEI WEN

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Chair of Committee,	Paul V. Gratz
Committee Members,	A. L. Narasimha Reddy
	Jiang Hu
	Samuel Palermo
	Duncan M. Walker
Head of Department,	Miroslav Begovic

December 2020

Major Subject: Computer Engineering

ABSTRACT

Current mobile applications have rapidly growing memory footprints, posing a great challenge for memory system design. Emerging non-volatile memory (NVM) has the potential to alleviate these issues due to its higher capacity per cost than DRAM and minimal static power. Meanwhile NVM has longer access latency compared to DRAM and limited write endurance. Therefore, integration of these new memory technologies in the memory hierarchy requires a fundamental rearchitecting of traditional system designs. In this work, we first propose a hardware-based memory manager (HMMU) that addresses both types of memory in a flat space address space. We design a set of data placement and data migration policies within this memory manager, to exploit both memory technology's advantages. Experiments show that the HMMU significantly reduces overall memory latency and energy consumption. However, we also found that hardware-only approaches have limited vision in time due to limited hardware resources on chip. To further improve HMMU performance, we integrate software information such as programmers' hints or application profiling, which reveal the longer-term memory access pattern and data object properties. Thus, our design combines the execution time advantage of pure hardware approaches integrated with data object properties in a global scope. Experiment results show that our design achieves a 40% reduction in energy consumption with only a 16% performance degradation versus the all-DRAM memory system.

In the HPC/Data domain, a primary problem is the interconnection network scalability, to service the ever-increasing number of nodes. Photonic-links is a promising technology to solve this problem: its higher bandwidth allows the router to connect more nodes, while the low signal loss makes long-distance links possible. Both factors help to reduce the average number of hops between nodes across the network, provisioning low latency communications in massive scale systems. However, interconnection network needs redesign to adopt the photonic links due to different physical and device properties. We first listed the design workflow for interconnection network and introduced a highly efficient event-driven simulator. Then we conducted a series of experiments

to explore the design space, and gave a quantitative comparison between interconnection networks built with pure electrical links and those with electronic/photonic hybrid design.

ACKNOWLEDGMENTS

First of all, I'd like to express the deepest gratitude to my advisor Prof. Paul V. Gratz. He lead me into the realm of research and guided me all the way to where I am today. During my pursuit of doctoral degree, he continuously inspired and supported me, with immense knowledge, professional expertise, motivation and patience. I learnt from him the appropriate attitude and approach to conduct researches.

I'm grateful to Prof. Narasimha Reddy, who devoted a lot of time to discuss with me and provide many constructive ideas. I'd also like to thank my committee members: Prof. Jiang Hu, Prof. Samuel Palermo, and Prof. Duncan M. Walker, for their encouragement and valuable comments.

My thank goes to Mian Qin, a perfect partner to collaborate with. I enjoyed the stimulating discussion, the struggle before deadline and all the time we worked together. Thanks to all the CAMSIN group members, for all the funs and moments we had in these years.

I'm extremely grateful to my parents and my wife Lindsey, for their unconditional love and dedication. I won't go through all those difficult times without their emotional support and encouragement. It's a great fortune to have you in my life.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a dissertation committee consisting of Professor Paul V. Gratz, Professor A. L. Narasimha Reddy, Professor Jiang Hu, and Professor Samuel Palermo of the Department of Electrical and Computer Engineering and Professor Duncan M. Walker of the Department of Computer Science and Engineering.

Chapter II and III were collaborated with Mian Qin of the Department of Electrical and Computer Engineering. Chapter IV was collaborated with Nic McDonald at Hewlett Packard Labs.

All other work conducted for the dissertation was completed by the student independently.

Funding Sources

Graduate study was supported by National Science Foundation, through grants I/UCRC-1439722 and FoMR-1823403, and generous support from DellEMC and Hewlett Packard Enterprise.

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

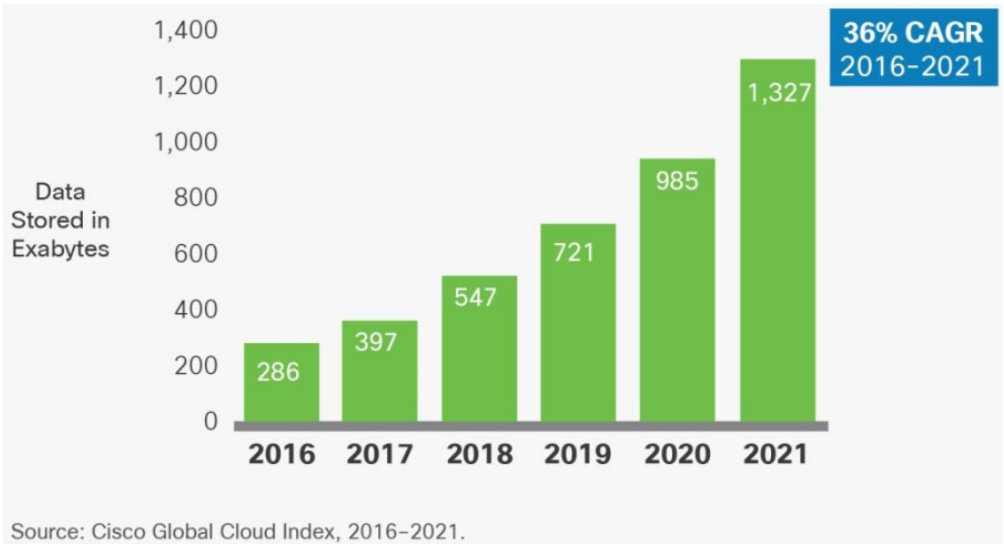
1. INTRODUCTION AND LITERATURE REVIEW

At the era of "big data", we've seen a tremendous growth of data size across the whole computing system spectrum, ranging from the edge device such as smartphones, all the way to the high-performance-computing (HPC) systems. Figure1.1 shows that both the stored data size and data traffic volume in data center are projected to expand at high annual growth rates. These trends pose great challenges on the memory system and the interconnection network. The same trend also appears in the mobile computing systems, as shown in Figure1.2.

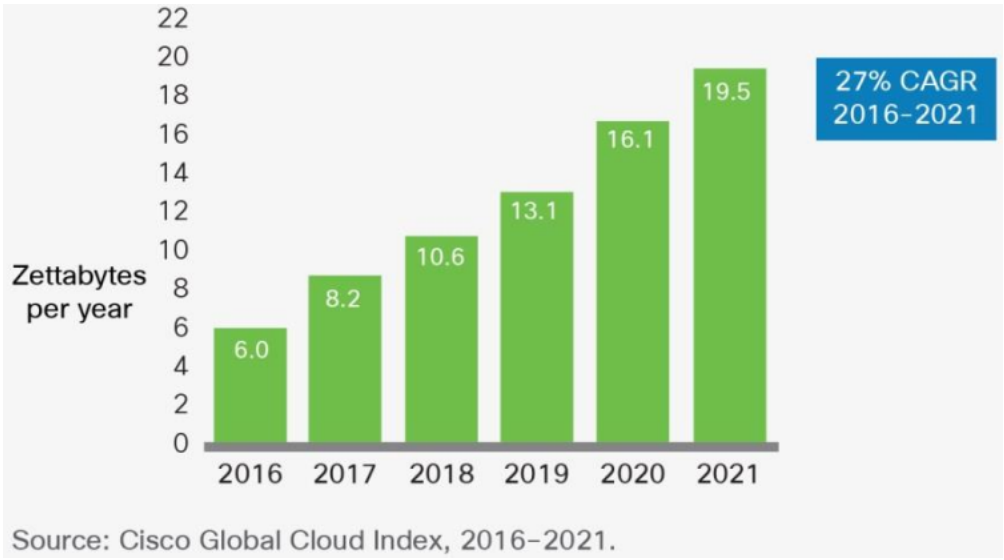
As the interconnection network is a major bottleneck of the HPC system [12], which decides the performance and energy efficiency of inter-node data movement, mobile computing systems care more about about the intra-node data management of the memory system. Energy efficiency, interconnection network and memory systems are also listed as the top 3 research challenges of building the exascale computing system [13]. The following sections briefly introduce these two topics.

1.1 Hybrid Memory for Mobile Computing System

As the demand for mobile computing power scales, mobile applications with ever-larger memory footprints are being developed, such as high-resolution video decoding, high-profile games, face recognition, speech-to-text conversion, natural language process, etc. This trend creates a great challenge for current memory and storage system design in these systems. The historical approach to address memory footprints larger than the DRAM available is for the OS to swap less used pages to storage, keeping higher locality pages in memory. Given the latencies of modern storage systems (even "high" performance SSDs [14, 15, 16]) are several orders of magnitude higher than DRAM, however, allowing any virtual memory swapping to storage implies incurring a severe slowdown. Thus mobile device rapidly expanded the DRAM size for the worst case possible memory footprint. For example, the DRAM capacity of the flagship phones from the Samsung Galaxy S series have expanded by 16X over the past ten years as shown in Figure 1.3



(a) Stored Data Growth Projection



(b) Data Traffic Growth Projection

Figure 1.1: Projection of Data Center Data Growth "Reprinted from [1]"

While this approach has been largely successful to date, the size of DRAM is constrained by both cost/economics and energy consumption. Unlike data centers, mobile devices are highly cost-sensitive and have a highly limited energy budget. Moreover, the DRAM technology has a substantial background power, constantly consuming energy even in idle due to its periodic refresh requirement, which scales with DRAM capacity. Therefore a larger DRAM means a higher power budget and a shorter battery life, particularly given recent hard DRAM VLSI scaling limits. The

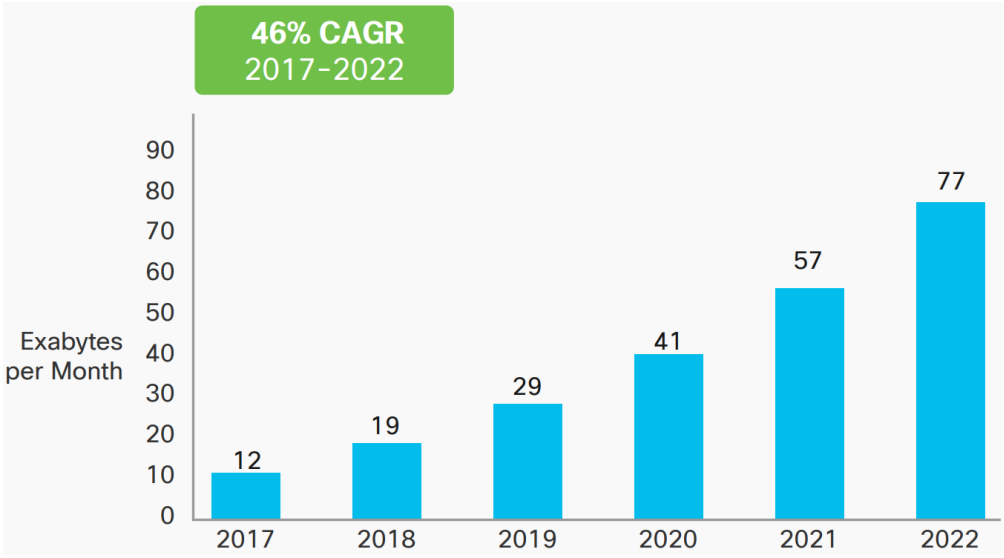


Figure 1.2: Mobile Data Traffic Growth"Reprinted from [2]"

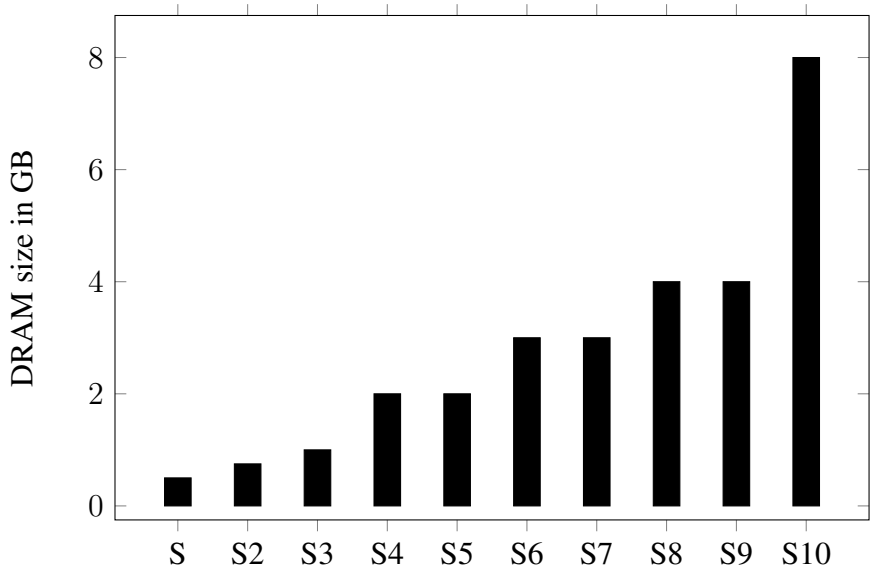


Figure 1.3: Samsung flagship smartphone DRAM size

approach of provisioning more DRAM is not sustainable and hard limits will soon be hit on the scaling of the future mobile memory system.

The emergence of several Non-Volatile-Memory (NVM) technologies, such as Intel 3D Xpoint [17], and memristor [18], and Phase-change-memory(PCM) [19], provides a new avenue to address this

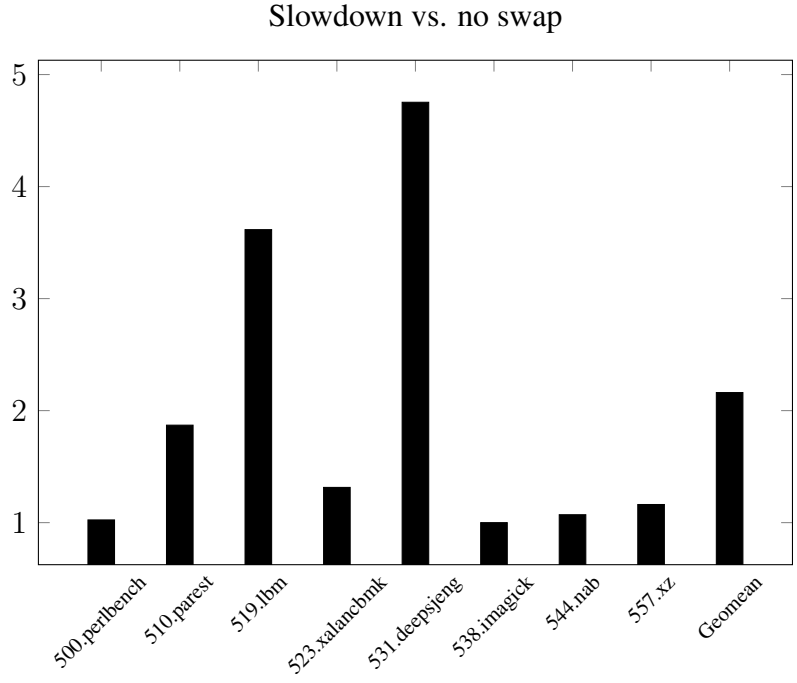


Figure 1.4: Performance impact of OS memory management.

growing problem. These new memory devices promise an order of magnitude higher density [20] per cost and lower static power consumption than traditional DRAM technologies, however, their access delay is significantly higher, typically also within one order of magnitude of DRAM. Further, these new technologies show significant overheads associated with writes and are non-volatile. Thus, these emerging memory technologies present a unique opportunity to address the problems of growing application workload footprints with hybrid memory systems composed of both DRAM and emerging NVM memories.

1.1.1 Hardware-Based Hybrid Memory Management

To exploit these new memory devices effectively, however, we must carefully consider their performance characteristics relative to existing points in the memory hierarchy. In particular, while memory access and movement in prior storage technologies such as flash and magnetic disk is slow enough that software management via the OS was feasible. With emerging NVM memory accesses at within an order of magnitude of DRAM, relying on traditional OS memory manage-

ment techniques for managing placement between DRAM and NVM is insufficient as illustrated in Figure 1.4.

In figure 1.4, a subset of benchmarks from the SPEC CPU2017 benchmark suite are executed in a system where around 128MB of the application's memory footprint is able to fit in the system DRAM directly. A ramdisk based swap file is set up to hold the remainder of the application memory footprint. Since this ramdisk swapfile is implemented in DRAM it represents an upper bound on the performance for pure software swapping, ie. without any added latency for NVM/storage access. The results shown are normalized against a system where sufficient DRAM is available to capture the entire memory footprint. As we see, in this arrangement, the cost of pure OS managed swapping to NVM would be quite high, with applications seeing an average of $\sim$2X slowdown versus baseline. As we will show, a significant fraction of this overhead comes explicitly from the costs of the required page fault handling.

Some existing work has begun to explore system design for emerging hybrid memories. Broadly this prior work falls into one of two categories, first, some advocate using DRAM as a pure hardware managed cache for NVM [21, 22]. This approach implies a high hardware cost for metadata management and imposes significant capacity and bandwidth constraints. Second, some have advocated for a purely software, OS managed approach [23, 24, 25]. As we discussed previously, this approach implies significant slowdowns due to software overhead of the operating system calls. In Chapter 2 we will present our solution: a new hardware managed hybrid memory management scheme which retains the performance benefits of caching, without the high metadata overhead such an approach implies. Compared to previous work.

1.1.2 Software/Hardware Cooperative Hybrid Memory Management

Let's summarise and compare the three categories of hybrid memory management we've discussed so far. First, using DRAM as a pure hardware managed cache for NVM [21, 22]. This approach implies a high hardware cost for metadata management and imposes significant capacity and bandwidth constraints. Second, purely software/OS managed approach [23, 24, 25]. As stated in last section, this approach implies significant slowdowns due to software overhead of the oper-

ating system calls. Third, the HMMU solution we proposed, which executes the data migration, under the guidance of a set data placement/migration policies implemented purely in hardware. Owing to the hardware efficiency, the HMMU is able to track the live memory requests in runtime and adapt to the change of memory access behaviors immediately. Being implemented in hardware with a limited state budget, however, also indicates its vision is limited to a short time window. Such limitation prevents it from capturing complex access patterns across a long range of time. Moreover, such a HMMU cannot tell the data object characteristics until observing several associated memory requests, thus it is incapable to decide the favorable memory device at the time of allocation, but rather must observe how the memory is used to determine where to best place it. As the consequence, data objects with mixed characteristics could end up sharing the same page and incur unnecessary page migration afterwards. Undesired page swaps waste substantial energy and exacerbate the write endurance of NVM device. Prior work by Berger *et al.* [26] attempt a profile-driven method optimizing data allocation. Generally, a given program's authors have a better understanding on the data structure and the manner of accesses. For example, will this data be revisited frequently after allocation? Will it be intensively read or written?

In Chapter 3, we describe a hardware/software system combining the benefits of HMMUs together with the deeper insights into memory usage that the programmer and profiling can provide. Our memory allocator allows the programmer to choose the preferable memory device. Such inputs are relayed to HMMU as a hint, to help it decide data placement/migrations. By incorporating such knowledge along with the HMMU, we significantly reduced the writes to NVM by 14% while the power consumption was 9% less than the prior HMMU solution.

1.2 Interconnection Network with Photonic Links

Interconnection network refers to the communication components used to exchange data between subsystems. Depending on the distance between the processing nodes, it could be categorized to Network-On-Chip(NoC), chiplet interconnect(chips are placed off-die but in the same package) or Inter-chip networks. As Dennard scaling came to the end, the growth of single processor was limited by the power dissipation. Hence, people turned to the solution of multiprocessors

(CMPs) interconnected via networks-on-chip (NoC), attempting to improve the overall system performance by workloads distribution among larger number of processing nodes. However, to fully exploit the hardware parallelism, multi-core architectures require higher inter-core and core-memory bandwidth as the system scales up. Besides the impact on performance, the bandwidth issue is also highly associated with the energy cost. As shown in the Figure 1.5, the energy consumption of data transfer increase rapidly along with the connection distance especially beyond the off-chip range.

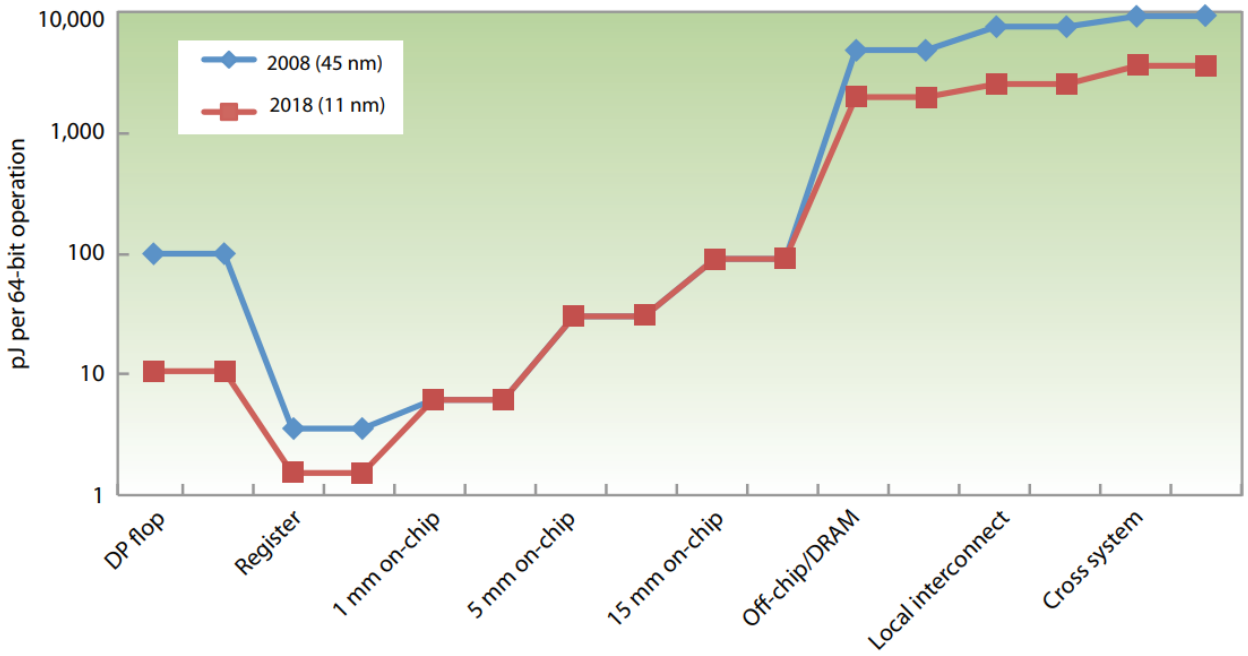


Figure 1.5: Energy Cost of Moving Data in Different Distances"Reprinted from [3]"

However, the electrical link bandwidth is constrained by its physical and electrical properties such as the number of pins, energy cost and the complexity of global wires.

Nano-photonic link is a promising technique to address the bandwidth and energy problems. It provides 10 —100x higher communication bandwidth than the traditional electrical link, while consumes significantly less energy. The power efficiency gain could reach 2 —3x on chip and 10 —30x off chip compared to electrical links. Furthermore, photonic links suffer significantly less

signal loss in long range propagation. Given the different features of photonic links, one will need to re-architect the interconnection network in order to exploit its technical advantages.

When it comes to interconnection architecture design, we need to consider a wide scope of parameters and options. This list includes but is not limited to: number of routers/nodes, bus protocol, fabrics, routing algorithms, network topology, router micro-architecture, arbitration mechanism. Moreover, these design choices are usually correlated hence it is impossible to investigate each decision point manually. In Chapter 4, I will introduce the Supersim [27], a simulator project that I worked on. It is designed in event-driven mechanism and supports simulation up to million-node scale network. I developed several new features and functions based on that framework, and conducted studies on two million-node scale network architectures.

1.3 Thesis Statement and Organization

This dissertation mainly presents our efforts to address two problems that emerged with the 'big data' trend: the memory pressure for mobile computing systems and the scale up of interconnection network for HPC/data centers. Chapter 2 and 3 discussed the hybrid memory system that we believe to be an optimal solution to the mobile computing memory problem. We created the HMMU which could place and migrate data under the guidance of memory access pattern. The HMMU has significant lower overheads as the data profiling and migration are all executed by hardware. In Chapter 3, we proposed a hardware/software co-operative approach for hybrid memory management that, which integrates the programmer's knowledge as hint information into the HMMU. This enables our system to recognize long-term access pattern as opposite to the pure hardware-based approach, it also helps the HMMU on choice of memory type at the time of allocation, before any memory references were recorded yet.

Chapter 4 showed my explorations in the photonic-link interconnection network design. Besides the specific architecture I worked on, I extended the functions of Supersim simulator and conducted simulations on several different interconnection network architectures. This provides a fast and reliable procedure to quickly sweep a vast space of parameter dimensions and compare their performances.

2. HARDWARE-BASED HYBRID MEMORY MANAGEMENT *

This chapter presents our solution to the increasing demand on mobile computing memory: the Hardware-Based Hybrid Memory Management(HMMU). In the first section we introduce the distinctive characteristic of Non-Volative Memory(NVM) and explain why the hybrid memory is deemed as the future of mobile computing memory systems. Then we list and compare the prior works on hybrid memory management from other groups. In the third and fourth sections, I illustrate the algorithms and mechanism of the HMMU, followed by the implementation details. In the last two sections, we evaluate our policies in several metrics and analyze how we achieve significant improvements compared to existing systems.

2.1 Nonvolatile Memory Technology Characteristics

With emerging non-volatile memory technologies providing more memory system capacity, density, and lower static power, they have the potential to meet the continuously increasing memory usage of mobile applications. Given their different characteristics from traditional DRAM and storage, however, the design of systems comprising these new technologies together with traditional DRAM and storage is an open question. Here we examine the characteristics of these new memory technologies and the existing proposals to date on how to leverage them in system designs. Table 2.1 shows the relative characteristics of several emerging non-volatile memory technologies against traditional DRAM and storage [28, 7, 8]. While HDD and Flash have 100k and 2k times larger read access latency than DRAM respectively, the emerging NVM technologies have read access latencies typically within one order of magnitude of DRAM. Meanwhile emerging non-volatile memory technologies provide higher memory system capacity, density and lower static power.

Given their different characteristics from traditional DRAM and storage, however, the design

*Parts of this chapter are adapted with permission from "Hardware Memory Management for Future Mobile Hybrid Memory Systems" by F. Wen, M. Qin, P.V. Gratz and A.L.N. Reddy, 2020. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Volume 39, 3627 - 3637, Copyright 2020 by IEEE.

Table 2.1: Approximate Performance Comparison of Different Memory Technologies[7, 8, 9]

Technology	HDD	FLASH	3D XPoint	DRAM	STT-RAM	MRAM
Read Latency	5ms	$100\mu s$	50 - 150ns	50ns	20ns	20ns
Write Latency	5ms	$100\mu s$	50 - 500ns	50ns	20ns	20ns
Endurance (Cycles)	$> 10^{15}$	10^4	10^9	$> 10^{16}$	$> 10^{16}$	$> 10^{15}$
$ per GB	0.025-0.5	0.25-0.83	6.5 [29]	5.3-8	N/A	N/A
Cell Size	N/A	$4 - 6F^2$	$4.5F^2$ [20]	$10F^2$	$6 - 20F^2$	$25F^2$

of systems comprising these new technologies together with traditional DRAM and storage is an open question. Here we examine the characteristics of these new memory technologies and the existing proposals to date on how to leverage them in system designs.

Further, we note that in these new technologies writes are often more expensive that reads both in terms of latency as shown and endurance/lifetime cost, as well as energy consumption for writing.

The relative closeness in performance and capacity to traditional DRAM of emerging NVM technologies argues for a different approach to memory management than traditional, OS or hardware-cache based approaches. In the remainder of this section, we examine the prior work approaches to the design of hybrid memory systems.

2.2 Prior Works on Hybrid Memory Management

2.2.1 Operating System-Based Memory Management

Hassan *et al.*, Fedorov *et al.* and propose to leverage the OS to manage placement and movement between NVM and DRAM [23, 24]. They treat NVM as a parallel memory device on the same level as that of DRAM in the memory hierarchy. They argue that this approach can yield better utilization of the large NVM capacity without wasting the also relatively large DRAM capacity. Their approach is similar to the traditional approach of using storage as a swap space to extend the DRAM main memory space. Direct application of this approach to NVM creates some difficulties, however. When a given requested data is found to be in the swap space on the NVM, a page fault occurs which must be handled by operating system. The latency of this action is

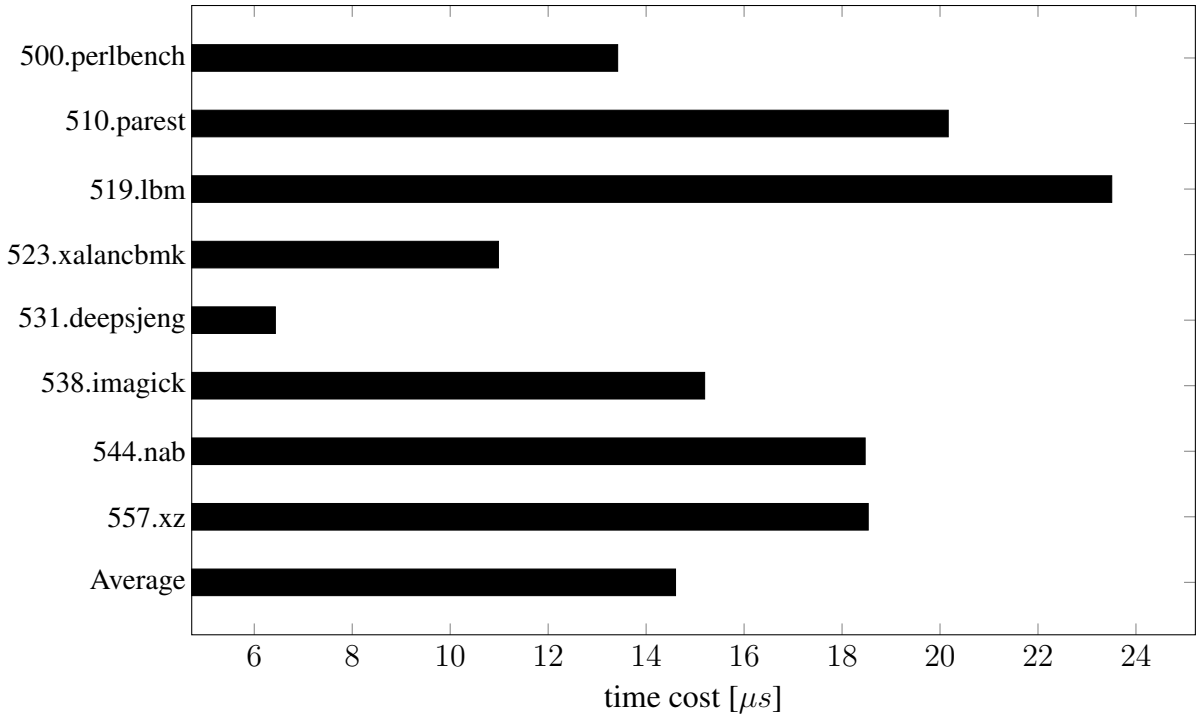


Figure 2.1: Page Fault Penalty Analysis

not only comprised of the device latency itself but also the induced OS context switch, and page fault handling. While in traditional storage systems with *ms*-level latencies, that cost is negligible, with the latency of SSD and other NVM devices significantly decreased, the OS management overheads come to dominate this latency. Figure 2.1 quantified the latency cost of the page fault system calls performed in the same experiment as mentioned in Figure 1.4. Here we collected the number of major page faults and the systime for each of the benchmarks to estimate the average cost per page fault (here we omitted minor page faults since it's overhead is relatively small, less than 0.5us according to our profiling). The figure shows the average major page fault cost to be $\sim 14.6us$. Compared to the data in Table 2.1 we see that this latency is quite close to the access delay of FLASH, and much slower than the access time of emerging NVM technologies. Thus, we expect that the operating system will become a significant bottleneck for this type of design. This implicates lots of software related overheads in the kernel swapping subsystems (page copying, page table manipulation, TLB/cache misses, etc.). With the performance gap closing between the

emerging NVM technologies and existing DRAM, SPAN [24] observes that the naive readahead prefetching in the OS swapping subsystem that optimizes traditional spinning disks introduce unnecessary latency to the OS page fault handling when fast, random accessible NVM is used as swapping area. SPAN leverages sophisticated learning based page prefetch technique to mitigate the OS swapping overhead for NVM device. However, the pure software implementation of SPAN still face the problem of prohibitive high cost of software overhead for page swapping which motivate us to turn to hardware oriented approaches to further reduce the latency for page migration between two tiers of memory.

2.2.2 Hardware-managed DRAM Caches and Related Approaches

Other groups have proposed using DRAM as the cache/buffer for NVM, and thus turning DRAM into the new last level cache[21]. Similar schemes have also been applied to other memory devices with latency discrepancy in heterogeneous-memory-system(HMS). For instance, 3D-stacked DRAM was proposed as a cache for off-chip DRAM in the works[30, 31, 32, 33]. A common theme in all these designs is the difficulty in lookup and maintenance of the tag storage, since the number of tags scales linearly with the cache size. Assuming the cache block size is 64B and 8 bytes of tag for each block, then a 16GB DRAM cache requires 2GB for the tag storage alone. That is much too large to fit in a fast, SRAM tag store. Much of the prior work explores mechanisms to shrink the tag storage overhead [34]. Some researchers explored tag reduction [35]. Others aimed to reconstruct the cache data structure. For instance, some works combine the tag or other meta-data bits into the data entry itself [31, 36].

Another issue these works attempt to address is the extended latency of tag access. DRAM devices have significantly greater access latency than SRAM. Additionally, their larger cache capacity requires a longer time for the tag comparison and data selection hardware. If the requested data address misses in the TLB, it takes two accesses to the DRAM before the data can be fetched. Lee *et al.* attempted to avoid the tag comparison stage entirely by setting the cache block size to equal the page size, and converting virtual addresses to cache addresses directly in a modified TLB [37]. This approach, however requires several major changes to the existing system architec-

ture including requiring extra information bits in the page table, modifying the TLB hardware and an additional global inverted page table.

Broadly, several issues exist with the previously proposed, hardware-based management techniques for future hybrid memory systems.

- As with traditional processor cache hierarchies, every memory request must go through the DRAM cache before accessing the NVM. Prior work shows that this approach is sub-optimal for systems where bandwidth is a constraint and where a parallel access path is available to both levels of memory [38]. Further, given the relatively slow DRAM access latency requiring a miss in the DRAM before accessing the NVM implies a significantly higher overall system latency.
- These works largely assume an inclusive style caching. Given the relative similarity in capacity between DRAM and NVM, this implies a significant loss of capacity.
- Given the capacities of DRAM and NVM versus SRAM used in processor caches, a traditional cache style arrangement implies a huge overhead in terms of cache meta-data. This overhead will add significant delays to the critical path of index search and tag comparison, impacting every data access.

Liu *et al.* propose a hardware/software, collaborative approach to address the overheads of pure software approaches without some of the drawbacks of pure hardware caching [39]. Their approach, however, requires modifications both to the processor architecture as well as the operating system kernel. These modifications have a high NRE cost and hence is difficult to be carried out in production.

2.2.3 HMMU Solution

We propose a hardware-based hybrid memory controller that is transparent to the user and as well as the operating system, thus it does not incur the overheads of management of OS based approaches. The controller is an independent module and compatible with existing hardware architectures and OSes. The controller manages both DRAM and NVM memories in flat address space to leverage the full capacity of both memory classes. Our approach also reserves a small

portion of the available DRAM space to use as a hardware-managed cache to leverage spacial locality patterns seen in real application workloads to reduce writes to the NVM. Compared to previous work, our project has the following advantages:

- With a ratio of 1/8 DRAM vs 7/8 NVM, we achieved 88% of the performance of an untenable full DRAM configuration, while reducing the energy consumption by 39%.
- Compared to inclusive DRAM caches, we preserve the full main memory capacity for the user applications.
- Parallel access to both the DRAM and NVM is supported, rendering a higher effective memory bandwidth. This also helps to suppress the excessive cache insertion/replacements and prevent cache thrashing.
- The data placement and migration are executed by hardware. This eliminates the long latency incurred by the OS managed virtual memory swap process.
- Memory management and allocation are performed with a combination of page and sub-page-block sizes to ensure the best utilization of the available DRAM and to reduce the number and impact of writes to the NVM.

2.3 HMMU Design

Here we describe the proposed design of our proposed hardware memory management for future hybrid memory systems. Based on the discussion in the last section and cognizant of the characteristics of emerging NVM technologies, we aim to design a system in which the latency overheads of OS memory management are avoided, while hardware tag and meta-data overheads of traditional caching schemes are minimized.

2.3.1 System Architecture Overview

Figure 2.2 shows the system architecture of our proposed scheme. The data access requests are received by the Hybrid memory management unit (HMMU), if they miss in the processor cache. These are processed based on the built-in data placement policies, and forwarded with address translation to either DRAM or NVM. The HMMU also manages the migration of data between

DRAM and NVM, by controlling the high-bandwidth DMA engine connecting the two types of memory devices.

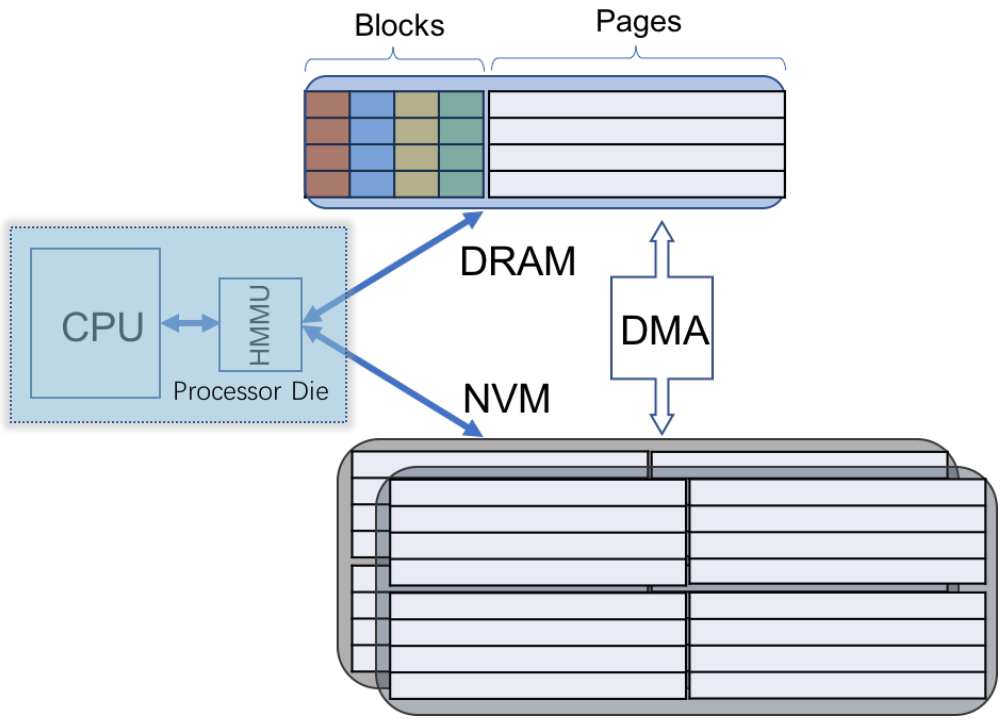


Figure 2.2: System Architecture Overview

2.3.2 Data management policy

A key component of the proposed HMMU design is its data management policy, *i.e.* the policy by which it decides where to place and when to move data between the different memory levels. Traditionally, in processor caches and elsewhere, cache blocks are managed with 64-byte lines and policies such as set-associative are used to decide what to replace upon the insertion of new lines into a given cache level. While this approach yields generally good performance results in processor caches, there are difficulties in adapting it for use in hybrid memories. As previously discussed in Section 2.2.2, for a hybrid memory system of 16GB comprised of 64-byte cache-lines, the tag store overhead would be an impractically large 2GB. Extending the block size up to 4KB to

match the OS page size would significantly reduce the overheads of the tag store, bringing it down to 4MB for a 16GB space. Since the host operating system primarily uses 4KB pages, using any larger size than 4KB for block management, however, risks moving a set of potentially unrelated pages together in a large block, with little, if any spatial locality between different pages in the block. This is particularly true because the addresses seen in the HMMU are "physical addresses", thus physically colocated pages may come from completely different applications, with no spatial relationship.[2] As we will discuss, however, even managing blocks on a page granularity will yield greater than optimal page movements between "fast" (DRAM) and "slow" (NVM) memory levels, due to the fact that only subsets of the page are ever touched in many applications. Thus, we will examine a hybrid scheme in which most of the fast memory is managed on a page basis, lowering tag overheads, while a small fraction is managed on a sub-page basis to reduce page movement when only small portions of each page are being used at a given time.

In terms of organization and replacement, using traditional processor cache policies of set associativity and LRU replacement become unwieldy for a memory system of this size. The practical implementation of such a set-associative cache requires either a wide/multi-ported tag array (which becomes untenable for large SRAM structures) or multiple cycles to retrieve and compare each way in the set sequentially. Prior work from the OS domain [40, 41] shows that, with a large number of pages to choose from, set associative, LRU replacement is not strictly necessary. Inspired by that, we first developed a simple counter-based page replacement policy.

2.3.2.1 Counter-based Page Management

Rather than implementing a set associative organization with the drawbacks described above, we instead propose to implement a secondary, page-level translation table internal to the HMMU as illustrated in Figure 2.3. The internal page table provides a one-to-one remapping, associating each CPU-side "physical" page number in the host address space to a unique page number in the hybrid memory address space, either in the fast or slow memory. Thus, any given host page can be

[2]While many systems do allow a subset of pages to be managed at larger granularities, the HMMU has no visibility to this OS-level mapping, thus we conservatively assume 4KB pages

mapped to any location in either fast or slow memory.

While this design gives great flexibility in mapping, when a slow memory page must be moved to fast memory (*i.e.* upon a slow memory reference we move that page to fast memory) it requires a mechanism by which to choose the fast memory page to be replaced. Inspired by prior work in the OS domain [40, 41], we designed the counter-based replacement policy for this purpose.

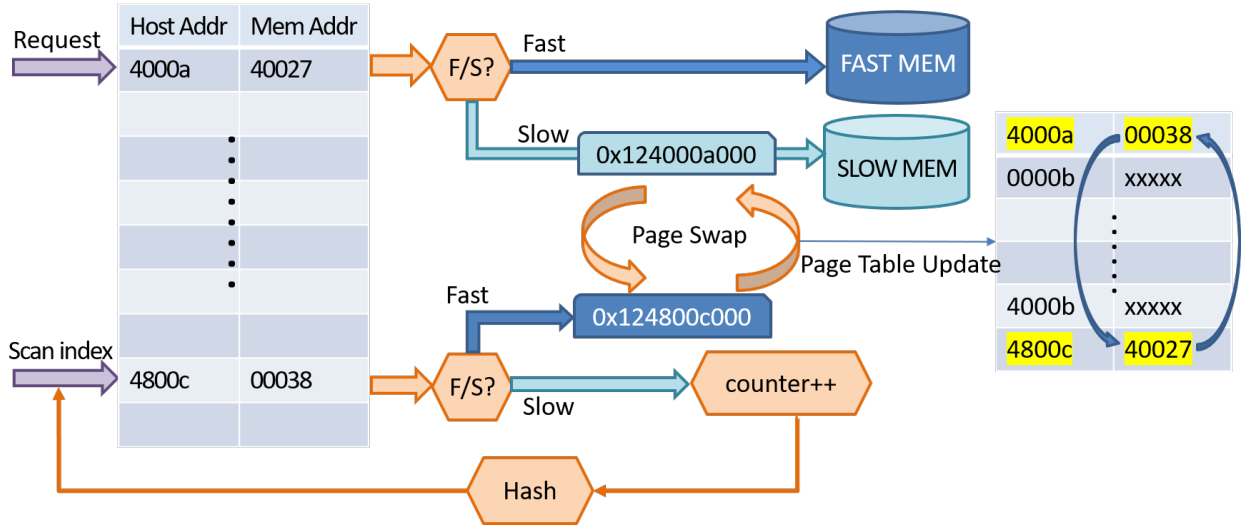


Figure 2.3: Counter-based Page Movement Policy

2.3.2.1.1 Algorithms and Design The counter-based replacement policy only requires one counter to keep track of the currently selected fast memory replacement candidate page, thus it has minimal resource overhead and can efficiently be updated each cycle. The chance that a recently accessed page gets replaced is very rare because 1. the total number of pages is very large; 2. the counter increases monotonically. To further reduce the possibility of evicting a recently touched page, however, we implemented a light-weight bloom filter that tracks the last 2048 accessed pages. Since checking against the bloom filter is parallel to normal page scan process, and is also executed in background, it adds no extra delay to data accesses. Algorithm 1 shows the details of the algorithm.

Algorithm 1: Counter-based Page Relocation

```
Function unsigned pgtb-lookup (address) is
    return page_table[address/page_size];
Function unsigned search-free-fast-page() is
    while pointed_page ∉ fast memory or
     pointed_page ∈ bloom filter do
        counter++;
        pointed_page  = pagetable[Hash(counter)];
    set candidate page as ready;
    return pointed_page;
Function counter-based-page-move(address) is
    pointed_page = pgtb-lookup (address);
    if pointed_page ∈ fast memory then
        directly forward the request to DRAM
    else
        if candidate page is available then
            initiate to swap the content between requested page and candidate page.;
            Call page-swap();
        else
            Forward the current request to NVM;
Function page-swap (source_page, target_page) is
    while page swap is not completed do
        if new requests conflict with pages on flight then
            Froward the requests to the corresponding device depending on the current
             moving progress
        Continue the page swap;
    Update the corresponding entries in page table.;
```

Figure 2.3 illustrates a simple example of this page movement policy. In this example memory address space, fast memory occupies internal page numbers 0 - 40000, while slow memory ranges from page number 40000 and beyond. In the figure, a request for host address 4000a arrives at the internal remapping page table. The corresponding internal page address in the memory address space is 40027, which in this case is the 28th page in the slow memory. Here we use a policy of page movement to fast memory upon any slow memory touch.[3] Thus, the HMMU directs the

[3]Note that the request is serviced immediately, directly from the slow memory, while the page swap happens in the background.

DMA engine to start swapping data between the requested page (40027) and the destination page in fast memory. Here, as described in Algorithm 1 the fast memory pages to be replaced is selected via the replacement counter, *i.e,* page 00038 in this example. Once the data swapping is completed, the memory controller updates the new memory addresses of the two swapped pages in the internal page table. Next, the counter searches for the next fast memory page replacement candidate. As the figure shows, the counter is passed through a hash function to generate an index into the internal page table. If the retrieved page number turns out to be in slow memory, the counter increments by one and the hash function generates a new index for the next query to the page table. Such process loops until it finds a page in the fast memory, which becomes the candidate destination for the page swapping.

Further details of the counter-based page management policy:

- Current requests are processed at top priority under all circumstances. Except for rare cases when a given write request conflicts with ongoing page movement, we always process the current request first. As for those rare cases, since all write requests are treated as non-blocking, the host system shall not suspend for them to complete. Therefore our design does not add overhead to the critical path of request processing.
- Due to the parallel nature of hardware, we search for free pages in fast memory in the background, without interference to host read request processing.
- Page-swap is initiated by the HMMU, however, it is executed by a separate DMA hardware module. Thus it does not impact other ongoing tasks.
- Data coherence and consistency are maintained during page movements.

We carefully designed the DMA process so that it could properly handle the new requests to the pages as they are being moved. All read requests and most write requests can proceed without blocking. In some very rare cases, the write requests are held until the current page copy finished.

2.3.2.2 Sub-page Block Management

Various applications could have widely different data access patterns: those with high spatial locality may access a large number of adjacent blocks of data; while others may have a larger stride

between the requested addresses. For applications with weak or no spatial locality, there is very limited benefit to moving the whole page of data into fast memory, as most of the non-touched data may not be used at all. Based on this observation, we propose a scheme for sub-page size block management, which manipulates the data placement and migration in finer granularity.

Algorithm 2: Sub-page Block Management

```
Function sub-page block management(address) is
    pointed_page = pgtb-lookup(address);
    if pointed_page ∈ fast memory then
        directly forward the request to DRAM
    else
        if the count of cached blocks > threshold value then
            if candidate free page available then
                initiate to swap the content between requested page and candidate page;
                Call page-swap();
            else
                Forward the current request to NVM;
        else
            initiate moving the block to cache zone
```

2.3.2.2.1 Data Migration Policy We set aside a small fraction of the fast memory and manage that area in a cache-like fashion with sub-page sized blocks. The basic algorithm used in shown in Algorithm 2. Upon the first accesses to a slow memory page, instead of moving the whole page into fast memory, we will only move the requested block of that page into the "cache" zone in fast memory. We then keep track of the total number of cached blocks belonging to every page. Only after the count of cached blocks meets a certain threshold will we swap the whole page to fast memory.

Figure 2.4 illustrates a simple example of the sub-page block relocation policy: The memory controller receives a request to host physical address 0x124000a200. In the first cycle, both the page table and cache metadata are checked in parallel, to decide the target memory device. If

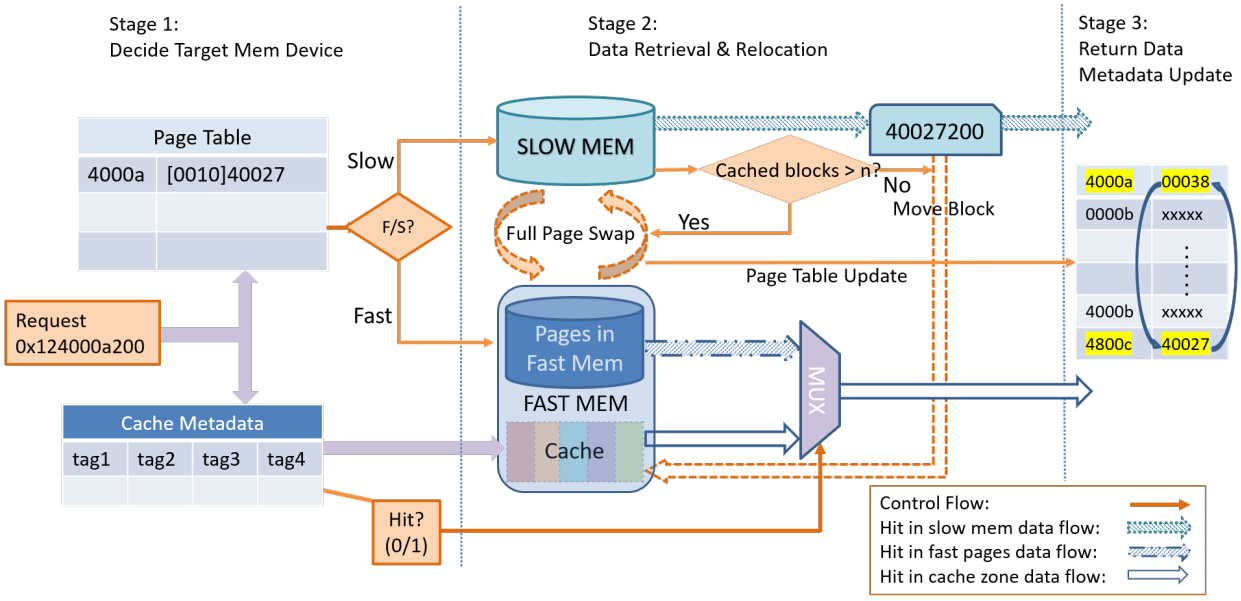


Figure 2.4: Sub-block Relocation Policy

the data is found only in the slow memory, the memory controller will trigger the data relocation process. The 4-bit counter in the page table entry tells the number of sub-page blocks that have been cached for the current page. Comparing the counter against the preset threshold, determines whether to start a full-page swap or a sub-block relocation. In the given example, the counter value is 2, which is smaller than the threshold value of 4. Thus only that specific block containing the requested data (0x40027200 to 0x4002727f) will be copied to the cache. It is possible that the data might be found in both slow memory and the cache at the same time. To enforce data consistency, we always direct the read/write request to the copy in cache. This dirty data will be written back to the slow memory upon eviction.

2.3.2.2.2 Fast Memory Cache Design As the page size is 4KB, we choose 128 bytes as a reasonable block size (this size also corresponds to the DRAM open page burst size, so it sees a significant performance boost versus other block sizes). The cache is organized as a 4-way associative cache. The cache uses a pseudo-LRU as the block replacement policy. We also enable a proactive cache recycling policy: when a block is accessed, if its underlying page is detected to have been relocated to fast memory, we would evict that block from the cache to save the space for

other blocks. Thus one block of data will not occupy the capacity of two copies in the fast memory at the same moment.

2.3.2.3 Hardware Cost and Overhead

Each page table entry takes $\log_2 \frac{\texttt{Memory Space}}{\texttt{Page Size}}$ bits to represent the page address. In addition, we need some bits for statistical meta-data such as the counter of misses occurring to the page. In our sample design, the memory space is 2GB and the page size is 4KB, thus the hardware cost per page could be rounded to $\log_2 \frac{\texttt{2GB}}{\texttt{4KB}} + 5\texttt{bits} = 3\texttt{bytes}$, and the total cost is 1.5MB. The page table cost scales linearly with memory size whereas the cost per entry only grows logarithmically. The meta-data for each cache set is comprised of three parts; four tags(8 bits $\times$ 4), pseudo-LRU bits (3) and dirty bits (4), which adds to 39 bits. The total cost is $39\texttt{bits} \times 2^{16} \approx 312\texttt{KB}$ Since the cache is read and check parallel to the access to the page table, there is no additional timing cost for handling regular requests. The DMA provides the non-conflict data relocation for sub-page block level as same as that of the page relocation.

2.3.2.4 Static versus Adaptive Caching Threshold

With both page and block migration available, a new question arises, how to choose wisely between these two policies for optimal results. We note that these policies have different characteristics as follows:

- With page-migration, the data is exclusively placed between NVM and DRAM device. Thus larger memory space is available to applications, and the bandwidth of both devices is available.
- Sub-page-block migration is done in an inclusive cache fashion, thus avoids the additional writes to NVM when the clean data blocks are evicted from DRAM.

For applications with strong spatial locality, whole page migration maximizes performance because the migration cost is only incurred once, and the following accesses hit in the fast memory. Alternately, sub-page block promotion benefits applications with less spacial locality, because it limits writes to NVM incurred by full page migration. We further note that application behavior

may vary over time with one policy being better in one phase and another better during another.

We therefore include in the page translation table an 8-bit, bitmap for tracking accesses to each sub-page block of the given page. This allows measurement of the utilization rate of promoted pages. If a large portion of blocks were revisited, then we lower the threshold to allow more whole page migration. Alternately, if few blocks were accessed we suppress the whole page promotion by raising the threshold value, decreasing the rate at which full pages are migrated.

2.3.2.5 Block Pre-fetch

The major benefits of data promotion comes from the shortened latency in future accesses. Hence we don't expect to see much improvement on the applications that have poor temporal localities, such as streaming applications. In these applications, data is rarely revisited after promotion, thus the only way to gain performance improvement is to exploit the spatial locality by pre-fetching the data before it is actually demanded. Prior work [42] showed that different pages sharing the same access patterns tend to have the same access patterns. In our design, we built a global table to record the probability of next demanded block for each access pattern within the page. We keep the three predictions with have the highest scores and will update the corresponding entry upon every memory access. An example is illustrated at 2.5 Excessive prefetches could cause cache pollution and excessive write backs to the NVM. To throttle prefetching, we implemented two dynamic mechanisms:

1. We monitor the prefetch accuracy by counting the number of prefetched blocks that were never used before eviction. When the accuracy fall below a certain value we raise the prefetch threshold score.

2. Prefetch only when free and clean block is available in current set.

2.4 HMMU System Evaluation

In this section, we present the evaluation of our proposed HMMU design. First, we present the experimental methodology. Then we discuss the performance results. Finally we analyze some of the more interesting data points.

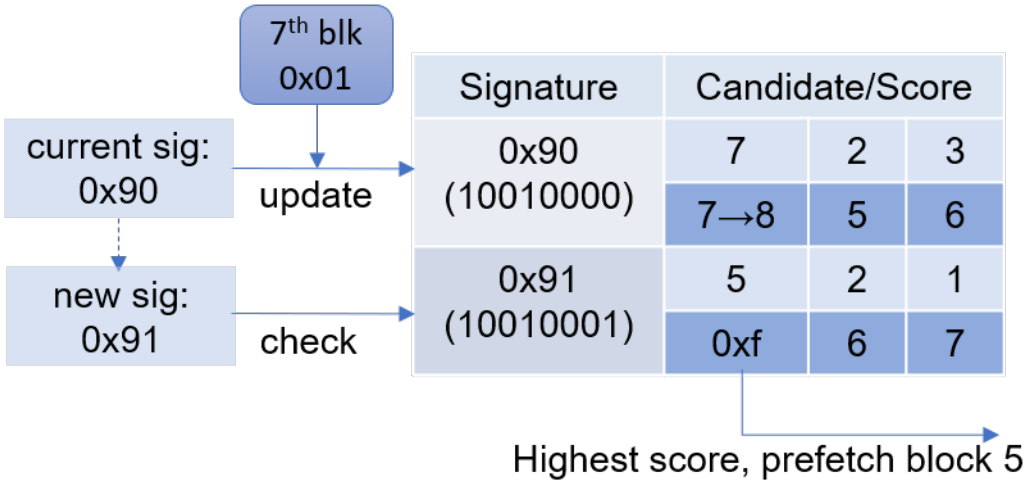


Figure 2.5: Prefetch Example

2.4.1 Methodology

2.4.1.1 Emulation Platform

Evaluating the proposed system presents several unique challenges because we aim to test the whole system stack, comprising not only the CPU, but also the memory controller, memory devices and the interconnections. Further, since this project involves hybrid memory, accurate modeling of DRAM is required. Much of the prior work in the processor memory domain relies upon software simulation as the primary evaluation framework with tools such as Champsim [43] and gem5 [44]. However, detailed software simulators capable of our goals impose huge simulation time slow-downs versus real hardware. Furthermore, there are often questions of the degree of fidelity of the outcome of arbitrary additions to software simulators [45].

Another alternative used by some prior work [24] is to use an existing hardware system to emulate the proposed work. This method could to some extent alleviate the overlong the simulation runtime, however, no existing system supports our proposed HMMU.

Thus, we elected to emulate the HMMU architecture on an FPGA platform. FPGAs provide flexibility to develop and test sophisticated memory management policies. The hardware-like nature of FPGA platform provides near-native simulation speed and the accuracy that is unattainable

for software simulators. The FPGA communicates with the ARM CortexA57 CPU via a high-speed PCI Express link, and manages the two memory modules(DRAM and NVM) directly. The DRAM and NVM memories are mapped to the physical memory space via the PCI BAR(Base Address Register) window. From the perspective of the CPU, they are rendered as available memory resource same as other regions of this unified space.

Our platform emulates various NVM access delays by adding stall cycles to the operations executed in FPGA to access external DRAM. The platform is not constrained to any specific type of NVM, but rather allows us to study and compare the behaviors across any arbitrary combinations of hybrid memories. In the following sections, we would show the simulation results with different memory devices. The detailed system specification is listed in Table 2.2.

Table 2.2: Emulation System Specification

Component	Description
CPU	ARM Cortex-A57 @ 2.0GHz, 8 cores, ARM v8 architecture
L1 I-Cache	48 KB instruction cache, 3-way set-associative
L1 D-Cache	32 KB data cache, 2-way set-associative
L2 Cache	1MB, 16-way associative, 64kB cache line size
Interconnection	PCI Express Gen3 (8.0 Gbps)
Memory	128MB DDR4 + 1GB NVM
OS	Linux version 4.1.8

We measured the round trip time in FPGA cycles to access external DRAM DIMM first, and then scaled the number of stalled cycles according to the speed ratio between DRAM and future NVM technologies, as described in Section 2.1. Thus we have one DRAM DIMM running at full speed and the other DRAM DIMM emulating the approximate speed of NVM Memory.

2.4.1.2 Workloads

We initially considered several mobile-specific benchmark suites, including the CoreMark [46] and AndEBench [47] from EEMBC. We found however that these suites are largely out of date and do not accurately represent the large application footprints found on modern mobile systems.

Also, in some cases they are only available as closed source [47] and thus are unusable in our infrastructure. Instead, we use applications from the recently released SPEC CPU 2017 benchmark suite [10]. To emulate memory intensive workloads for future mobile space, we selected only those SPEC CPU 2017 benchmarks which require a larger working set than the fast memory size in our system. The details of tested benchmarks are listed in Table 2.3.

To ensure that application memory was allocated to the HMMU's memory, the default Linux malloc functions are replaced with a customized jemalloc [48]. Thus the HMMU memory access was transparent to the CPU and cache, and no benchmark changes were needed.

Table 2.3: Tested Workloads[10]

Benchmark	Description	Memory footprint
Integer Application		
500.perlbench	Perl interpreter	202MB
520.omnetpp	Discrete Event simulation - computer network	241MB
523.xalancbmk	XML to HTML conversion via XSLT	481MB
531.deepsjeng	Artificial Intelligence: alpha-beta tree search (Chess)	700MB
557.xz	General data compression	727MB
Float Point Application		
510.parest	Biomedical imaging: optical tomography with finite elements	413MB
519.lbm	Fluid dynamics	410MB
538.imagick	Image Manipulation	287MB
544.nab	Molecular Dynamics	147MB

2.4.1.3 Designs Under Test

Here we test the following data management policies developed for use with our HMMU:

- **Static:** A baseline policy in which host requested pages are randomly assigned to fast and slow memory. This serves as a nominal, worst-case, memory performance.
- **PageMove:** The whole 128MB DRAM is manged on the granularity of 4k pages. When a memory request is missed in fast memory, the DMA engine will trigger a page relocation from slow memory to fast memory, as described in Section 2.3.2.1.

- **StatComb:** Here 16MB out of the 128MB DRAM is reserved for sub-page block relocation, managed in the cache-like fashion, as described in Section 2.3.2.2. The remainder of the DRAM is managed on a full page basis. An empirically derived static threshold of 4 blocks touched is used to determine when a full page should be moved to the page portion of DRAM.
- **AdpComb:** Same as StatComb, except that, as described in Section 2.3.2.4, an adaptive threshold is used to determine when the full page should be moved.
- **AllDRAM:** Here we implement a baseline policy in which there is sufficient fast memory to serve all pages in the system and no page movement is required. This serves as a nominal, best-case but impractical memory performance design.

2.4.2 Results

2.4.2.1 Energy Saving

Emerging NVM consumes minimal standby power, which could help save energy consumption on mobile computation. We evaluated and compared the energy spent in running SPEC 2017 benchmarks between the full DRAM configuration and our policies. We referred to Micron DDR4 technical spec [49] for DRAM and recent work on 3DxPoint [50] for NVM device power consumption, respectively (Table 2.4).

Table 2.4: Power Consumption of DDR4 and 3D-XPoint

Technology	DDR4	3Dxpoint
Read Latency	50ns	100ns
Write Latency	50ns	300ns
Read Energy	4.2nJ	1.28nJ
Write Energy	3.5nJ	8.7nJ
Background Power	30mW/GB	~ 0

We normalize the energy consumption of our policies to that of the AllDRAM configuration and present them in the figure 2.6. In the figure we see that all three techniques save a substantial

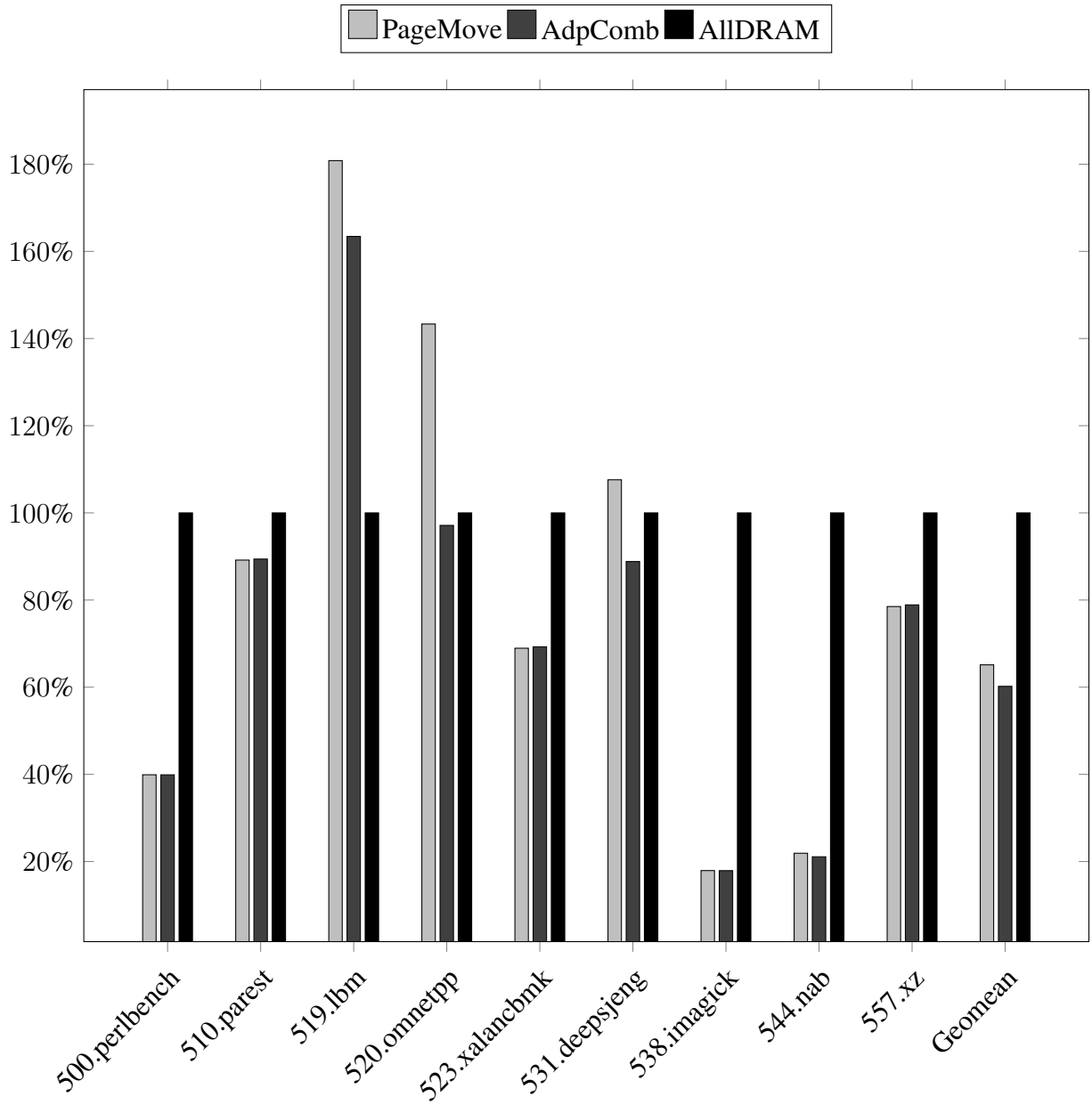


Figure 2.6: Energy Consumption Comparison

amount of energy. On average the AdpComb adaptive policy only consumes 60.2% energy as compared to the AllDRAM configuration, while the PageMove and StatComb policies are at 65.1% and 63.6%, respectively. That said, several benchmarks see energy consumption increases under the PageMove policy, while StatComb, sees a significant regression in energy consumption for 519.lbm. AdpComb, while also seeing increased energy consumption under 519.lbm, shows better

energy consumption than the other two policies for nearly all cases.

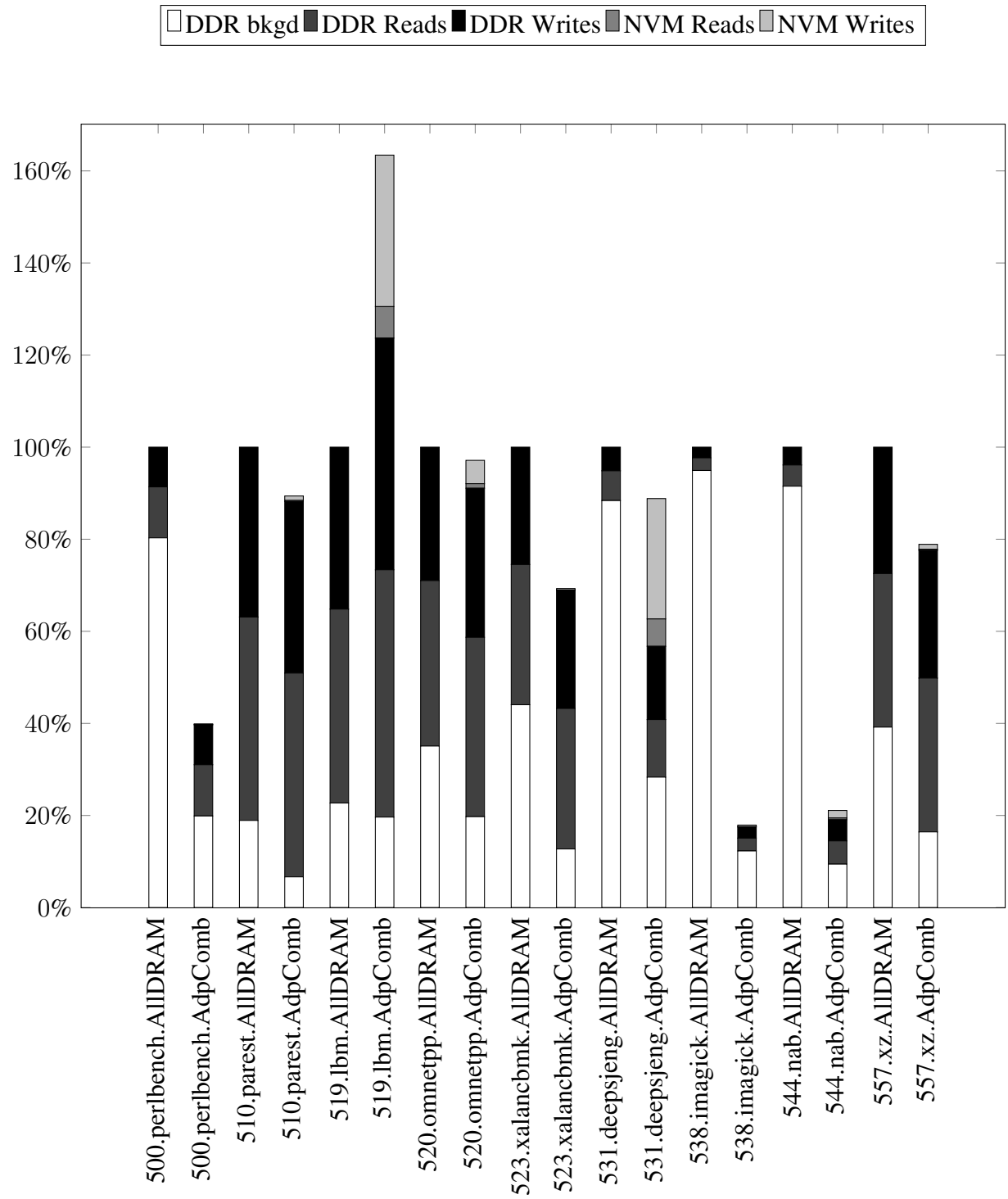


Figure 2.7: Energy Consumption Breakdown

Further investigating the distribution of energy consumption, we track the DRAM background power, number of DRAM read/writes and NVM read/writes. We present the comparison between AdpComb and AllDRAM in Figure 2.7. Since 7/8 of the memory was replaced with NVM, the standby power shrinks significantly. Although write operations to NVM dissipate more energy than DRAM, the AdpComb policy avoids most of this increase by absorbing many writes in DRAM. Our policies saw the greatest energy efficiency improvement with applications imagick and nab, which spent 17.9% and 21.1% energy compared to full DRAM. We find that these two applications have high processor cache hit rates and spent most time in computation. Thus they have few references to the memory, and the largest portion of energy was spent on DRAM background power. Thus AptComb policy's advantage of having much lower DRAM static power is best exploited. Our policies did pretty well with all benchmark applications except lbm, which spent 63% more energy. This application incurred a massive number of cache block writebacks to NVM. We investigated the case and found lbm has the highest percentage of store instructions among all benchmark applications [51]. This creates many dirty blocks, and thus writebacks are expected when blocks are later evicted. The amount of writes is also amplified by the writebacks of cache blocks.

2.4.2.2 Runtime Performance

Figure 2.8 shows the speedup attained by the different designs under test for the various benchmarks in the SPEC CPU 2017 benchmark suite. Here all the results are normalized to the runtime of the ideal, AllDRAM, DRAM configuration. We see that the average performance of AdpComb is 88.4%, while the random static allocation "Static" only yields 40% of the full DRAM performance. Thus, the adaptive policy achieves more than 2x performance benefit versus the worst-case, static allocation policy under the same memory resource. Generally the AdpComb policy outperforms the other two policies we propose, though interestingly, for many benchmarks, including perlbench, parest, xalancbmk, xz, imagick, and nab, PageMove comes within 5% of the performance of AllDRAM.

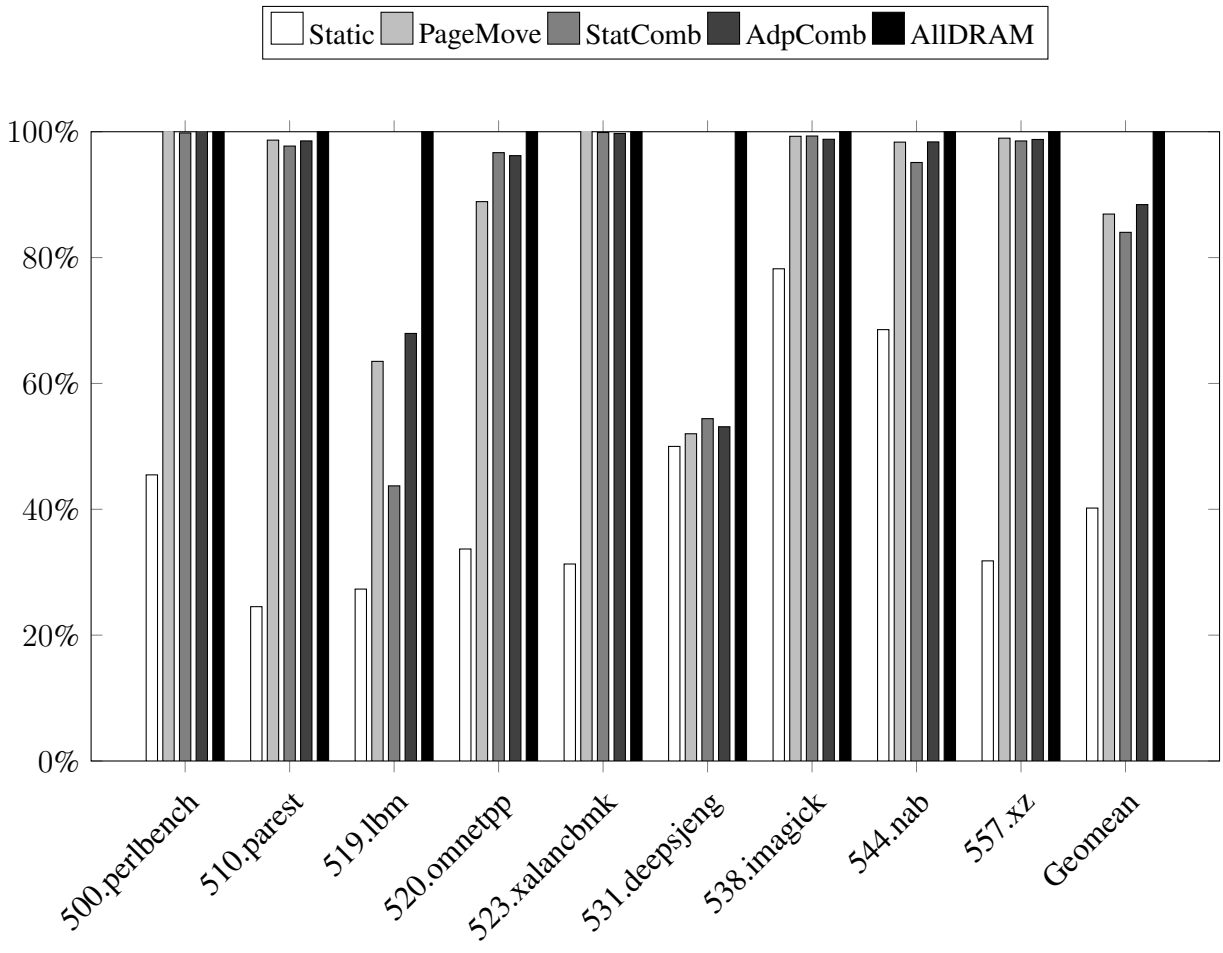


Figure 2.8: SPEC 2017 Performance Speedup

2.4.3 Analysis and Discussion

The adaptive AdpComb policy successfully reduces energy by 40%, with a modest 12% loss of the performance versus an unrealistic and unscalable AllDRAM design. AdpComb attempts to make the optimal choice between the PageMove and the StatComb block migration policy. In the remainder of this text, we will further analyze the experiment results.

2.4.3.1 PageMove Policy Performance

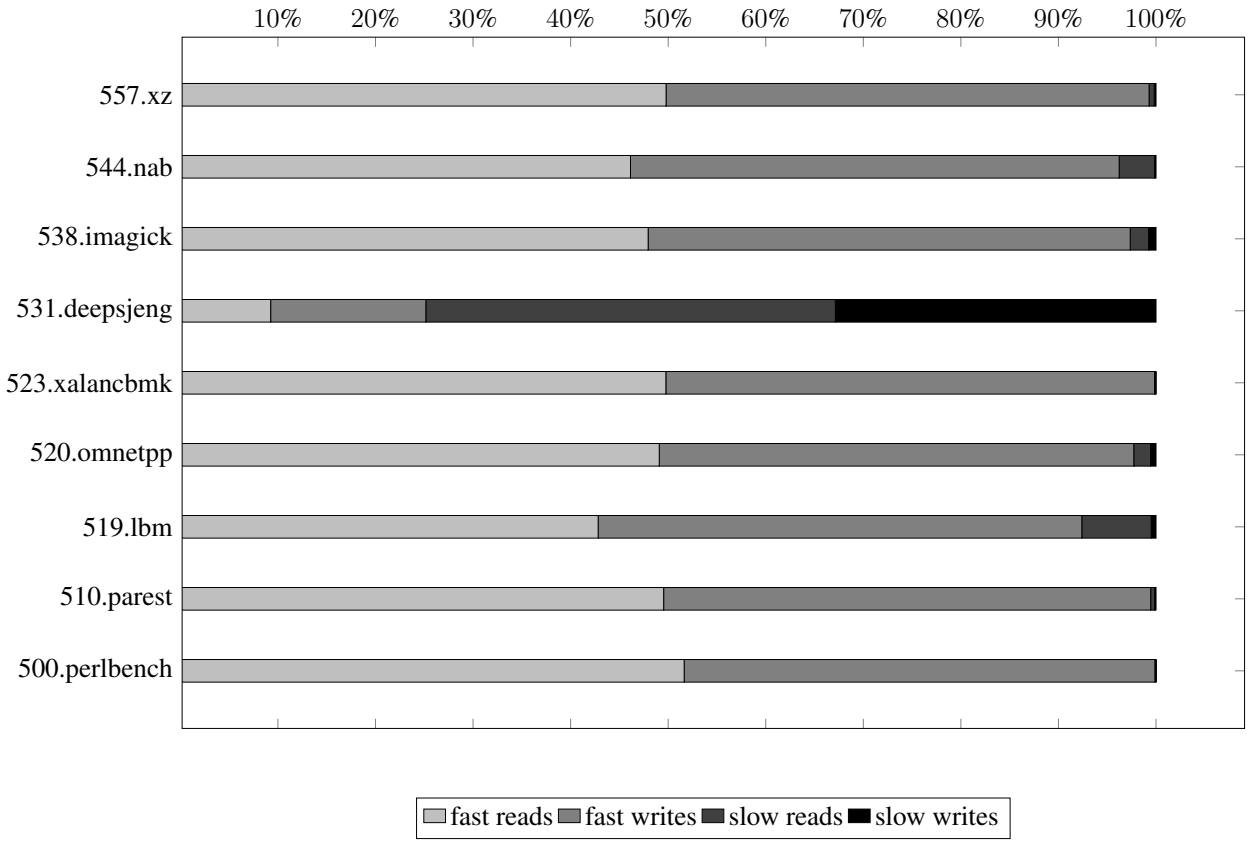


Figure 2.9: Memory Accesses Breakdown of PageMove Policy

The PageMove policy has similar average runtime performance (86.9%) to the adaptive AdpComb policy (88.4%). Figure 2.9 shows the breakdown of memory requests that hit in the fast pages and slow pages respectively. When compared to the speedup in Figure 2.8, we see the bench-

marks which PageMove policy works best have most of their memory requests hitting in the fast pages, while the hit rate in slow pages become negligible. This provides a large performance boost considering that the system's slow memory is 8x slower than the fast memory.

In the figure, the StatComb policy has an overall speedup of 84% against the AllDRAM configuration. The difference is mainly contributed by 519.lbm and 544.nab. As we will show, however, StatComb does still provide significant benefits in terms of total writes to NVM.

The PageMove policy performs worst on the benchmark 531.deepsjeng, with a slowdown of 52% versus AllDRAM. We divided the number of hits in fast memory by the occurrences of page relocation, and found that deepsjeng has the lowest rate (0.03) across all the benchmark applications (Geomean is 3.96). This suggests that when a page is relocated from slow memory to fast memory, the remainder of that page is often not extensively utilized. Further, we also see an exceptionally high ratio of blocks moved to cache versus page relocation. The geometric mean of all benchmarks is 10.5 while deepsjeng marks 397. This is a sign that in most cases, the page is only visited for one or two lines, and never accumulates enough cached blocks to begin a whole page relocation. To sum up, deepsjeng has a sparse and wide-range memory access pattern, which is quite difficult to prefetch effective data or improve performance.

519.lbm presents another interesting case, since its performance is also poor. Similar to deepsjeng, the hit rate in fast memory is low in contrast to the number of page relocations. However, a key difference is that over 60% of the cached blocks were evicted after its underlying pages relocated to fast memory. This indicates that lbm walks through many blocks of the same page and triggers the whole page relocation quickly. On that account, we deduce that this benchmark will benefit from a configuration with more fast pages and a smaller cache zone. We reran this benchmark with a cache size of 8MB and the threshold value of 1, and found a supportive result of 8% performance gain on top of the default threshold value of 4.

2.4.3.2 Writes Reduction and NVM lifetime Saving

Unlike the traditional DRAM, emerging NVM technologies have different characteristics for reads and writes. Write operations dissipates more than 8x the energy of reads [7]. Moreover,

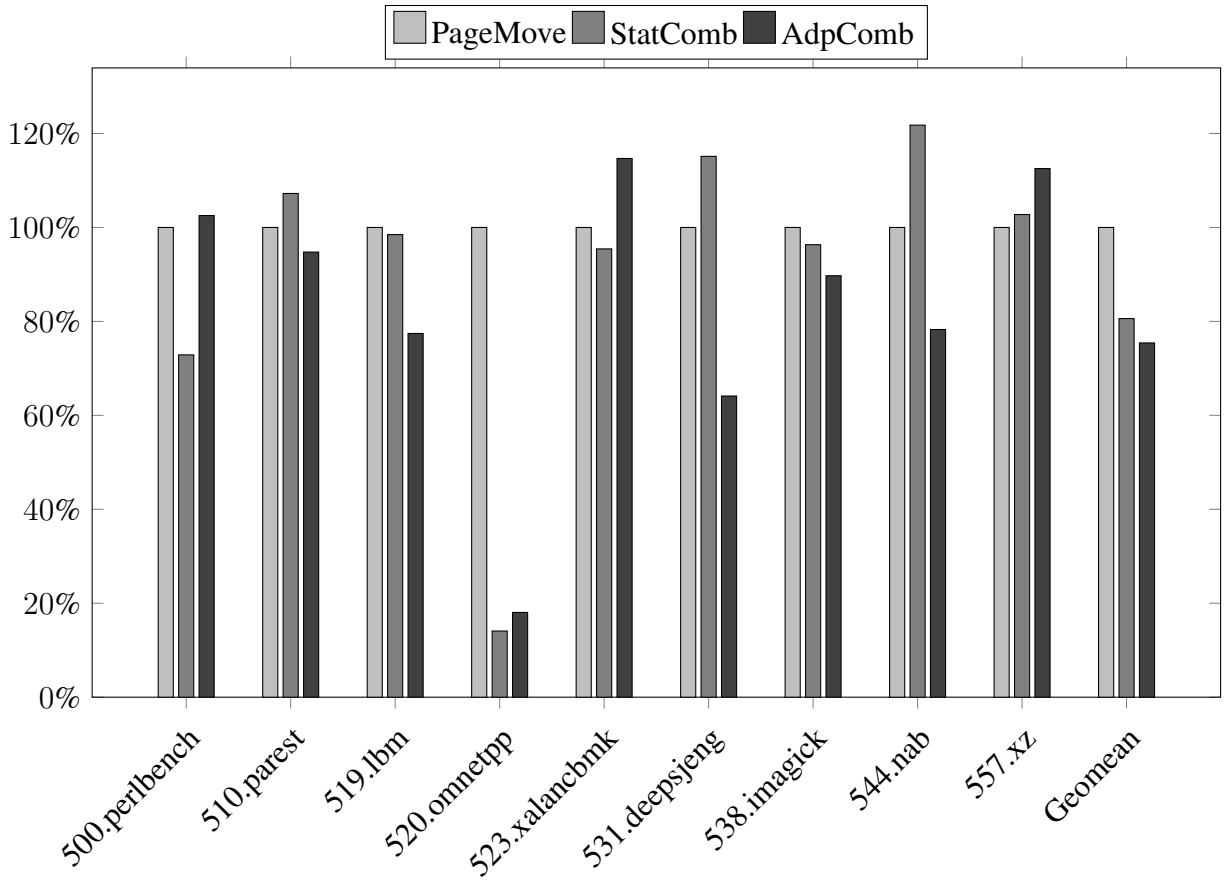


Figure 2.10: Writes to NVM

NVM technologies often have limited write endurance, i.e, the maximum cycles of writes before they wear out. Hence, if we could reduce the amount of writes, we could greatly save energy consumption and extend the lifetime of NVM device. Figure 2.10 shows the percentage of writes to slow memory for both techniques, normalized against the number of writes seen in the PageMove policy. Please note that we measure not only the direct writes from the host but also the writes induced by page movements and sub-page block writebacks to slow memory. In the figure we see that our combined policy has an average of 20% fewer writes than the PageMove policy. While several benchmarks benefit from the sub-page block cache, this advantage is strongest with omnetpp, with a drop of 86%. The detailed analysis of this particular benchmark is presented in the next section.

2.4.3.3 *Sensitivity to Threshold*

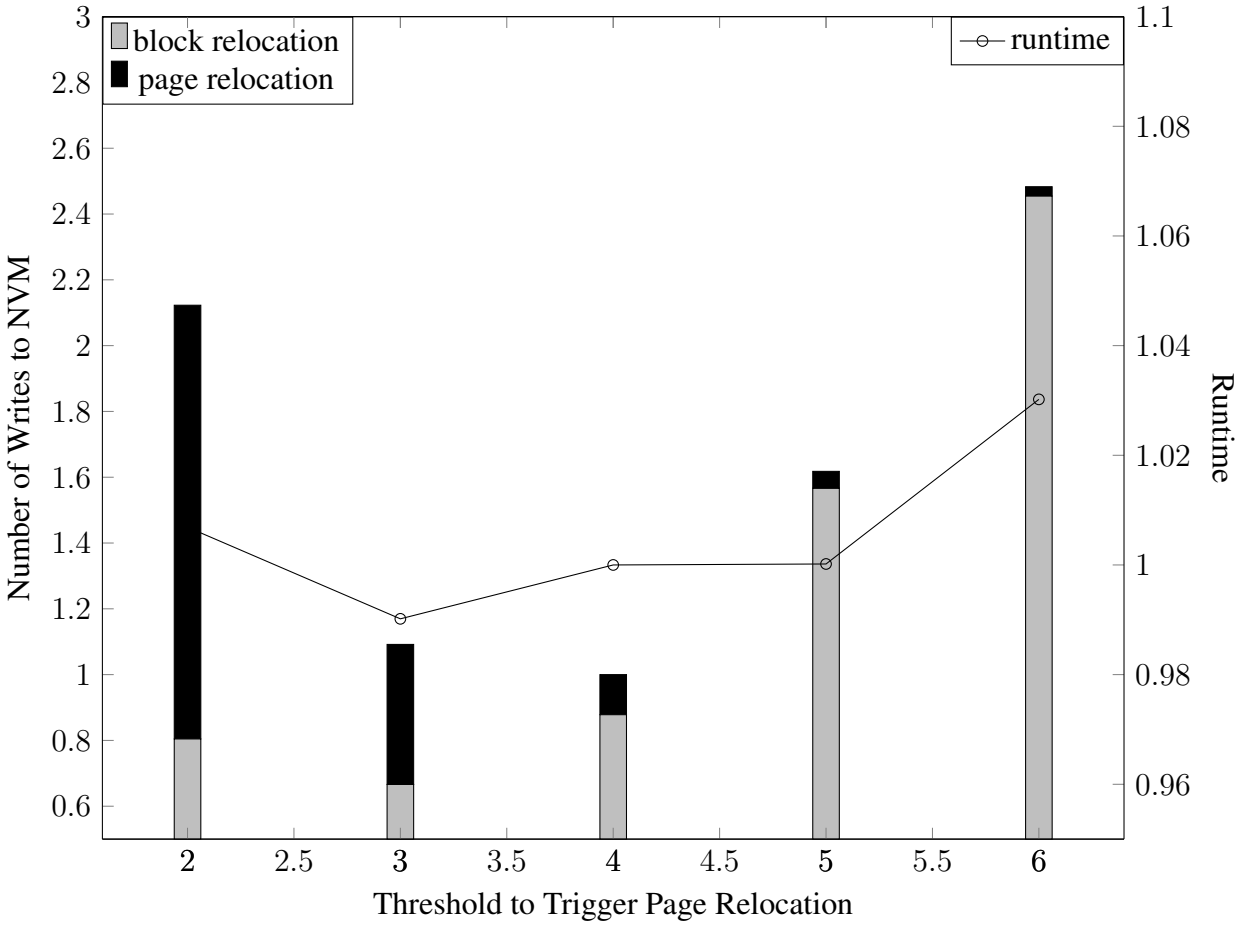


Figure 2.11: Omnetpp Performance Analysis

The extraordinary reduction of writes for omnetpp is intriguing. We reran the tests with different StatComb static page relocation thresholds and examined the changes in run time and total numbers of writes to NVM. In Figure 2.11, we normalized all numbers to the value for a threshold of 4, the threshold used in StatComb. The runtime varied according to the same trend as the number of writes, and the threshold value of 4 turned out to be the overall sweet spot. Both metrics started to deteriorate rapidly when the threshold value shifted. Then we measured the number of writes to NVM incurred by page relocation and block relocation, respectively. The results represented by

stacked bars, reveals the reason why threshold of 4 is the best choice: More pages were relocated when the threshold was lowered. On the other hand, the amount of block migration grew rapidly as the threshold increased. The trade-offs reached perfect balance at the value of 4, which had a slightly more page moves than that of value 5, yet significantly fewer block migrations.

2.4.3.4 Adaptive Policy

The analysis above showed that the whole page promotion policy favors certain benchmark applications, in which most blocks were revisited on the promoted pages. Meanwhile other applications benefit from sub-page block promotions as only a subset of blocks were re-utilized. If we could always choose the correct policy for each application, then we could expect the optimal results for overall performance. These results reinforce the reasoning behind our AdpComb policy's adaptive threshold, wherein for applications where pages are mostly utilized full page movement is completed quickly, while for applications where accesses are sparse, page movement is postponed till most of the page has been touched once.

2.5 Summary

A wide spectrum of non-volatile memory (NVM) technologies are emerging, including phase-change memories (PCM), memristor, and 3D XPoint. These technologies look particularly appealing for inclusion in the mobile computing memory hierarchy. While NVM provides higher capacity and less static power consumption, than traditional DRAM, its access latency and write costs remain problematic. Integration of these new memory technologies in the mobile memory hierarchy requires a fundamental rearchitecting of traditional system designs. Here we presented a hardware-accelerated memory manager that addresses both types of memory in a flat space address space. We also designed a set of data placement and data migration policies within this memory manager, such that we may exploit the advantages of each memory technology. While the page move policy provided good performance, adding a sub-page-block caching policy helps to reduce writes to NVM and save energy. On top of these two fundamental policies, we built an adaptive policy that intelligently chooses between them, according to the various phases of the running

application. Experimental results show that our adaptive policy can significantly reduce power consumption by almost 40%. With only a small fraction of the system memory implemented in DRAM, the overall system performance comes within 12% of the full DRAM configuration, which is more than 2X the performance of random allocation of NVM and DRAM. By reducing the number of writes to NVM, our policy also helps to extend device lifetime. Last but not least, our hybrid memory solution also has a great economic advantage over the traditional DRAM memory system, as the cost per GB of NVM is significantly lower than DRAM as shown in Table 2.1.

3. SOFTWARE/HARDWARE COOPERATIVE HYBRID MEMORY MANAGEMENT

Based on the HMMU hardware unit stated in the last chapter, we present in this chapter the hardware/software cooperative solution that combines programmer's knowledge with the hardware profiling information. The first section demonstrates how user's knowledge could help hybrid memory management. Then we retrospect the function of each component and the way they coordinate with each other, to present the readers a picture of the whole system. The third section illustrates the algorithms and mechanism of the user hint integration, followed by the detailed implementation explained in the fourth section. In the evaluation section, we conducted a series of experiments on the hw/sw cooperation solution and compared its performance to prior works. The last section discusses the experiment results and analyzes the source of achieved gains.

3.1 Background and Motivation

Table 2.4 shows the various characteristics of DRAM and NVM, highlighting which would best host data objects with different access patterns. From the discussion in the last chapter, frequently read data should be stored on DRAM for its shorter access delay, along with more intensively written objects due to high NVM write overheads and limited endurance. As for the large-size data with rare visits, they should be placed on NVM for its high capacity and low static power consumption. Programmer's domain knowledge helps to better understand the program memory access pattern [52, 53, 54]. Thus, programs with user hints can achieve better efficiency on heterogeneous memory, and data profiling [55, 56, 54, 57] can also be an auxiliary support to reach the correct decision of data placement and migration. In the following sections, we will discuss some related works for programmer hints, data profiling and practical software/hardware approaches for data migration and how prior works motivate us to better leverage software/hardware approaches to exploit heterogeneous memory.

3.1.1 User-Hint Based

With a comprehensive understanding of the program, user/programmer knowledge is shown to be crucial for better exploiting heterogeneous memory. Based on system level heuristic information, Mogul [52] suggests migrating operating system cold pages based on page types, file types to the slow memory. Meswani *et al.* proposed TLM Malloc [53] to allow programmers explicitly allocate memory buffers on different class of memories. Dulloor *et al.* [54] developed a runtime profiling framework to efficiently place program data structures in heterogeneous memory.

In order to leverage programmer knowledge, we propose a hardware/software memory allocator that adopts programmer hints of data objects behavior, to improve the efficiency of data placement and migration. The scheme has an API that allows user to specify the preferred memory device for the requested data objects. The memory allocator hardware takes this into account in initial placement and subsequent data migration between DRAM and NVM to avoid unnecessary data movements, reducing writes and improving performance.

3.1.2 Data Profiling

Besides proactive program level user hints, runtime data profiling can be a complementary approach to accurately recognize data structures properties by analyzing the memory accesses activities during application execution. Profiling can enable learning of several key features such as access frequencies, spatial locality within or across pages, read/write ratio, etc. In several prior works, the built-in hot/cold page lists of the Linux were employed in the profiling algorithms [55]. A variety of algorithms derived from the CLOCK have also been proposed [58]. Others used the performance counters to identify the memory access patterns of different program phases [56]. These methods could only provide a system-level general evaluation, however, which lacks the fine grained information for each page.

In some proposals, activities of each page is tracked by inserting metadata bits into page table entries as profiling counters [59]. These counters, however, have very limited width, as the overheads grows linearly along with the memory capacity. Due to the high overheads, online profiling

approaches are constrained to monitor memory access behaviors in a relatively small time window. As the consequence, it is unable to detect complex memory access patterns that span across a long range of time. Moreover, since the profiling only starts after pages were allocated and referenced, the data objects are initially, obliviously placed on memory device during allocation. Thus, items with opposite access patterns, say heavily read/written, could end up sharing the same page, which could exhibit mixed behaviors and become difficult to profile. In the best case it still takes the profiler several rounds before it gained enough information to decide on a page migration. All these page migration could have been avoided if they were placed in the correct memory type in the beginning. Therefore we propose to integrate user-hints in the allocation policy.

3.1.3 Data Migration

Efficient data migration is critical for heterogeneous memory after we identify features for different data structures through user knowledge or data profiling. Prior works proposed both software and hardware approaches for efficient data migration. Generally, software data migration approaches require extension of the OS kernel functions [59, 60, 61, 55, 58, 52] or runtime/helper threads [56]. These OS-driven page migrations incur significant overheads: the process is usually triggered by a system interrupt, followed by context switch and kernel interrupt handling. A TLB shootdown might also be necessary after the page mapping was updated. The associated delays, which was once negligible compared to traditional storage media such as spinning disks, now become the dominant factors as the NVM device has much shorter access latencies. Researchers have tried different ways to mitigate the overheads of page-migration: Wu *et al.* constrained the page migration to only happen at the end of each phase of the application [56]. Yan *et al.* chose to move a bundle of pages at one time to amortize the cost of OS intervention. The hardware-based solutions typically use DRAM as inclusive cache for NVM device. There are two common problems with this solution: the tag size is enormous as NVM capacity is much larger than DRAM, and the additional access latency since every request must go through DRAM first. Many groups attempted to address the first problem by either shrinking the tag size [34, 35] or reorganizing the tag/data-entry structure [31, 36]. However, all these methods require significant modifications

to the page table or TLB, which render themselves less practical. Another caveat against using DRAM as cache of NVM is that the total available memory becomes less as the DRAM space is invisible to user space.

One recent work [62] successfully solved both of these issues via a hardware memory management unit (HMMU), which addresses the two memory devices in one flat address space, executing the page migration transparently behind a translation layer visible only to the HMMU. In the whole process OS doesn't need to step in and no changes were made to the TLB or the OS page table. In our design, we propose to combine user hints (either through domain knowledge or data profiling) interface with HMMU data migration scheme to better guide objects level data placement and migration on heterogeneous memory.

3.2 Design

Our approach here is to enable the adoption of the user's knowledge about data object reference patterns in hints that the system software sends to the HMMU, to improve the accuracy of data placement decision. Thus we propose a hardware/software co-operative architecture that is capable of both online/offline data profiling to execute data migration efficiently and effectively.

3.2.1 System Architecture Overview

Figure 2.2 shows the complete system architecture. Users' hints are taken during the data object allocation. By default, our memory allocator groups objects of a given type (*ie.* high writes, performance, low writes, etc.) and assigns them into a given page (page S1 in the fig.) based upon special mark bits which denote the type of memory desired. This information is then conveyed to the HMMU during memory page mapping. The HMMU makes data placement decision based on a set of comprehensive policies under different scenarios. When no free page is available, the HMMU scans the metadata of the internal page table for the cold pages(page S2 in the figure) pending to be swapped out from DRAM. After mapping finished, the memory requests are then forwarded to the allocated frames on the specific type of memory device. Besides the injection of user hints, the HMMU still maintains its data management functions as described by Wen *et*

al.[62]. A brief introduction of the HMMU and how it interacts with our user-hints information is presented in the later sections.

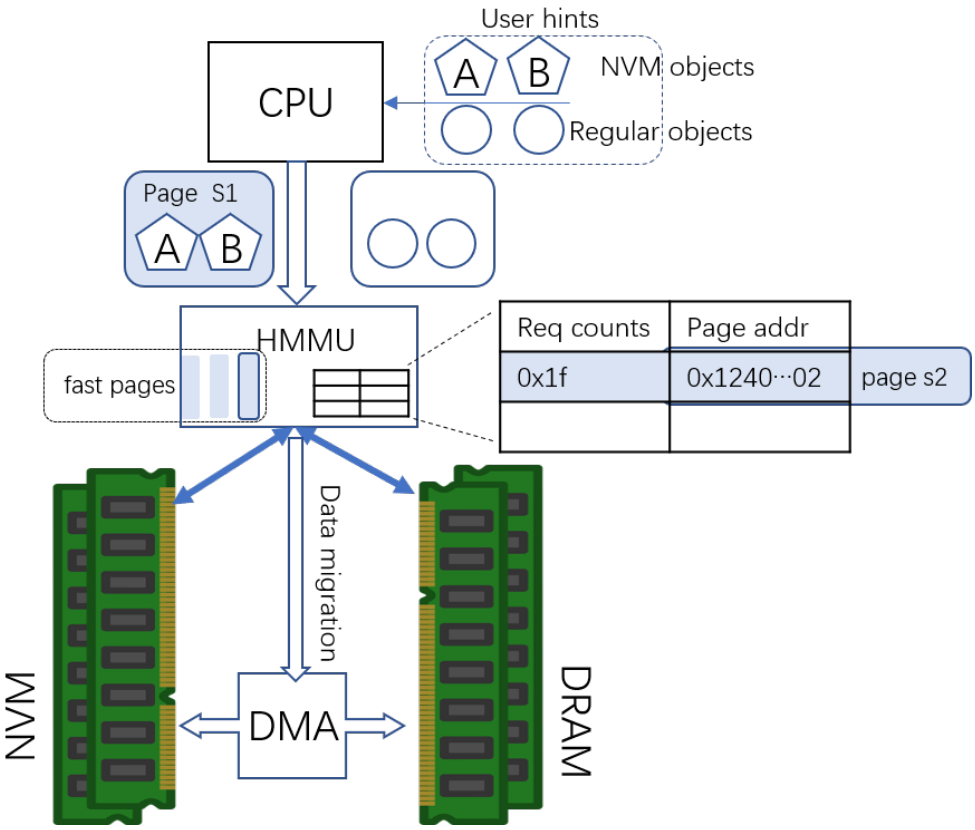


Figure 3.1: System Architecture Overview

3.2.2 Memory Allocator API

To accept users' hints about data objects properties, we created a series of memory allocation APIs that allow the programmer specify the desired memory type for the pertinent data objects. As the creators of programs, they often have deeper understanding about the memory access patterns of the key data structures, which helps to form better data management strategies beyond the HMMU's scope. As discussed in Section 3.1.2, the HMMU is incapable of detecting complex access patterns, as its limited hardware resource only supports monitoring of short-term behavior. With the help of user hints, we could identify the data objects that has reoccurring accesses between

intervals. Such data should be preferentially kept in the DRAM. HMMU profiles data objects by observing the received memory requests, thus it carries no relevant information at the time of allocation. Such oblivious allocation often places mixed data types together, which has negative impact on the data management throughout the object lifetime. Now that user hints are provided during allocation, we can group similar data into the same page and avoid unnecessary potential data migration. Our memory allocator is an extension of memkind by Intel [63]. It is compatible with the default glibc malloc function. Here is the API of memory allocation:

```
void* malloc(struct memkind *kind=DEFAULT, size_t size);
void* calloc(struct memkind *kind=DEFAULT, size_t num, size_t
    size);
void* realloc(void* ptr, size_t size);
void* free(void* ptr);
```

The programmer could either label the particular data object for NVM device, or leave the flag blank, allowing the allocator to handle it with default settings.

3.2.3 Baseline HMMU

The user hints collected from the software stack is forwarded to a hardware memory management unit (HMMU) inspired from prior work by Wen *et al.* [62]. As in the prior work, our HMMU is a memory controller unit that receives the memory requests from the CPU and addresses NVM and DRAM device in one flat memory space. It has an internal HMMU page table that translates the incoming physical address to the mapped memory frame on the memory device[1]. It keeps track of the recent activities of each page with metadata bits on the page table entry. It has a comprehensive policy of both data placement and migration. It has a built-in DMA engine that could directly move data between two memory device, without blocking the incoming requests from the CPU. These operations are all hidden from the OS and are executed highly efficiently in hardware.

[1]The HMMU's page table is orthogonal from the OS page table and invisible to it. In this system the OS only sees a large, flat physical memory space.

3.2.3.1 Page Swap

The HMMU tracks the number of references to each page in the last epoch. Once the number grows above the threshold value, a whole page migration is triggered. The threshold value is also learnt online to cope with phase changes of the running application. In the background, the HMMU scans the entire DRAM space with a pseudo-random pointer for choosing the target page to be migrated. It also records the most recently accessed pages in a bloom filter to preserve locality, protecting them from being swapped out.

3.2.3.2 Cache Partition

Despite the majority of DRAM space being available to user applications in a exclusive fashion, a small portion is reserved to be managed in an inclusive cache-like fashion. The reasoning is that whole page migration is unnecessary if few blocks are revisited. Under such circumstance, only the specific blocks needed are moved to DRAM, to save the time/energy cost of data movement.

3.2.3.3 Adaptive Threshold

Wen *et al.* [62] observed that data locality could vary starkly across different phases of application. Thus they set aside several metrics to evaluate the efficiency of page swap and block promotion. They calculate the average number of references to the most recently visited pages. They also have a few sampled sets to reflect the fluctuation of hits in cache zone. When these numbers increase, the threshold is lowered to allow more data moves. The threshold is raised to suppress migration between memories to save time/energy when these numbers decrease. This is commonly seen in streaming applications which traverse a vast range of data only once.

3.2.4 Data Management Policy

When the HMMU receives the first memory request to a "high writes" flagged page, for example, it assigns a free page in DRAM if available. Otherwise, the requested address is translated to the currently mapped memory frame. It also sets a special bit for this page in the HMMU's page table, which will trigger the page migration to DRAM next time it gets referenced.

Since DRAM pages are a limited resource, mechanisms are required to ensure that pages which are marked "high writes" are not locked into DRAM if in fact they are not being written often, or if DRAM pressure is otherwise high. After the page is mapped or migrated to the memory frame in DRAM, the HMMU sets a counter in the metadata bits of the corresponding page table entry. During every refresh epoch, the counter decrements by 1 if no write requests are received. When the counter decreases to zero, the page loses its special status and now becomes a normal page that is susceptible to being swapped back to NVM. The detailed algorithms is listed as below:

Algorithm 3: User-hints Data Placement/Migration

```
Function struct* pgtb_entry Lookup_pgtb(address) is
    struct* pgtb_entry=PageTable[address/page_size]; return pgtb_entry;
Function unsigned page-map (address, flag) is
    if !flag then
        forward the request to currently mapped memory frame;
    else
        if free DRAM page available then
            forward the request to the free DRAM page; update the page table with the
              page in DRAM
        else
            forward the request to currently mapped frame; set the special bit in the page
              table entry;
Function void access-page(address, is_write) is
    struct* pgtb_entry=Lookup_pgtb(address); if pgtb_entry.device is NVM then
        Trigger page migration to DRAM;
    else
        if is_write then
            pgtb_entry.write_counter=full_score;
        else
            if Refresh Epoch Ends then
                pgtb_entry.write_counter–
    if pgtb_entry.write_counter=0 then
        pgtb_entry.special_bit=0;
```

3.2.5 Hardware/Software Coordination

Since the programmers/users are not necessarily always aware of the actual memory access patterns of all their data objects, especially after cache filtering, their preferences of memory type are only passed to HMMU as hints, rather than the determinant of the data management. Thus, the system is free to ignore some or all of these hints in the event that the requested DRAM space is larger than the actual DRAM size. Throughout the different phases of applications, if the number of references or writes declines, the data object will "expire" and will be swapped back to NVM, even though it was marked as "high writes" or "performance" by the user when allocated. This coordination between HMMU and the software stack enables the design to have both long-term and short-term scope in data management.

3.2.6 Adaptive Throttling of Data Migration

When pages are marked with user hints, they have a higher priority in competition for the limited DRAM pages. These demands add stress on the memory system and may lead to thrashing among the other pages, after the memory footprint grows to a certain extent. Frequent page swaps incurs waste of memory bandwidth and energy, obviating any benefit had by leveraging the hints. Therefore we need to find a new way to throttle the data migration.

A typical scenario is in streaming applications, which traverse a vast range of addresses without recurrence. To deal with such applications lack of locality, we designed the adaptive throttling mechanism on our data migration policy. For the whole page migration, the HMMU counts the references to the page after it being migrated to DRAM, in each refresh epoch. If the average number of the last 128 accessed DRAM pages is larger than that of last refresh epoch, we lower the bar of whole page migration. Such applications have strong spacial locality and it's worthwhile to perform whole page migration, as the following access all hit in DRAM with shorter latency. Alternately, if the page was referenced less frequently than before, we raise the threshold to suppress page migration.

In the DRAM's cache partition (see Section 3.2.3, we sample 16 sets for accumulated refer-

ences per refresh epoch after each new block is inserted/replaced. We coordinate throttling the block migration until the optimal result is obtained.

3.2.7 Hardware Cost and Overhead

We note that the overheads are effectively the same between the HMMU [62] and the proposed HMMU user hints scheme. Due to careful flag bit multiplexing, the internal HMMU page table does not require any extra space.

3.3 Evaluation

In this section, we present the evaluation of our proposed hardware/software cooperative memory management solution for hybrid memory systems. First, we present the experimental methodology. Then we discuss the performance results. Finally we analyze some of the more interesting data points.

3.3.1 Methodology

3.3.1.1 Emulation Platform

Evaluating the proposed system presents several unique challenges because we aim to test the whole system stack, comprising not only the CPU, but also the memory controller, memory devices and the interconnections. Further, since this project involves hybrid memory, accurate modeling of DRAM is required. Much of the prior work in the processor memory domain relies upon software simulation as the primary evaluation framework with tools such as Champsim [43] and gem5 [44]. However, detailed software simulators capable of our goals impose huge simulation time slowdowns versus real hardware. Furthermore, there are often questions of the degree of fidelity of the outcome of arbitrary additions to software simulators [45].

Another alternative used by some prior work [24] is to use an existing hardware system to emulate the proposed work. This method could to some extent alleviate the overlong simulation runtime, however, no existing system supports the baseline HMMU [62], which we extend in this project.

Thus, we elected to emulate the HMMU architecture on an FPGA platform. FPGAs provide

flexibility to develop and test sophisticated memory management policies while its hardware-like nature provides near-native simulation speed. The FPGA communicates with the ARM CortexA57 CPU via a high-speed PCI Express link, and manages the two memory modules(DRAM and NVM) directly. The DRAM and NVM memories are mapped to the physical memory space via the PCI BAR(Base Address Register) window. From the perspective of the CPU, they are rendered as available memory resource same as other regions of this unified space.

Our platform emulates various NVM access delays by adding stall cycles to the operations executed in FPGA to access external DRAM. The platform is not constrained to any specific type of NVM, but rather allows us to study and compare the behaviors across any arbitrary combinations of hybrid memories. In the following sections, we would show the simulation results with different memory devices. The detailed system specification is listed in Table 2.2.

Table 3.1: Emulation System Specification

Component	Description
CPU	ARM Cortex-A57 @ 2.0GHz, 8 cores, ARM v8 architecture
L1 I-Cache	48 KB instruction cache, 3-way set-associative
L1 D-Cache	32 KB data cache, 2-way set-associative
L2 Cache	1MB, 16-way associative, 64kB cache line size
Interconnection	PCI Express Gen3 (8.0 Gbps)
Memory	128MB DDR4 + 1GB NVM
OS	Linux version 4.1.8

We measured the round trip time in FPGA cycles to access external DRAM DIMM first, and then scaled the number of stall cycles according to the speed ratio between DRAM and future NVM technologies, as described in the table 2.4. Thus we have one DRAM DIMM running at full speed and the other DRAM DIMM emulating the approximate speed of NVM Memory.

3.3.1.2 Approximating User-Hints through Code Profiling

Our solution is designed to enable programmer hints on data object allocation. As we are not the programmers for the various workloads examined, we instead profiled the benchmark with

valgrind [64] to determine how their data structures use memory, with two major metrics: the read/write ratio and the access frequency. We examined the data objects with the ten highest number of accesses, and selected those with the highest ratio of written bytes to read bytes. These data structures were flagged at malloc time as described in Section 3.2.2. Note, this somewhat naïve form of hinting likely leaves significant gains on the table versus having the actual programmer generate the memory hints.

We implemented the new malloc functions as an extension Intel's memkind project [63], which is a jemalloc-like memory allocator library. We also changed the underlying memory mapping functions to ensure that the application memory was allocated to the HMMU's memory. All these modifications are hidden behind the API, so minimal changes are needed in the benchmark source code.

3.3.1.3 Workloads

We use applications from the recently released SPEC CPU 2017 benchmark suite [10]. To emulate memory intensive workloads for future mobile space, we selected only those SPEC CPU 2017 benchmarks which require a larger working set than the fast memory size in our system. To diversify the workloads we also added a few benchmark applications from the parsec benchmark suites. The details of tested benchmark applications are listed in Table 2.3.

Table 3.2: Tested Workloads[10, 11]

Benchmark	Description	Memory footprint
SEPC 2017		
557.xz	General data compression	727MB
510.parest	Biomedical imaging: optical tomography with finite elements	413MB
538.imagick	Image Manipulation	287MB
PARSEC		
blackscholes	Option pricing with Black-Scholes Partial Differential Equation	610MB
facesim	Simulates the motions of a human face	298MB
freqmine	Frequent itemset mining	624MB
streamcluster	Online clustering of an input stream	219MB
ocean	Large-scale ocean movements simulation (HPC)	222MB

3.3.1.4 Designs Under Test

Here we test the following data management policies developed for use with our HMMU:

- **Static:** A baseline policy in which host requested pages are randomly assigned to fast and slow memory. This serves as a nominal, worst-case, memory performance.
- **Static-UserHints:** The data objects are allocated to the memory type specified by user's hints, and stay at the allocated memory frame until they are freed.
- **HMMU-only [62]** HMMU as proposed by Wen *et al.* [62], 16MB out of the 128MB DRAM is reserved for sub-page block relocation, managed in the cache-like fashion, as described in Section 3.2.3. The remainder of the DRAM is managed on a full page basis.
- **HMMU-UserHints:** Our proposed hardware/software cooperative data management policy. The baseline HMMU dynamic policies extended to incorporate the users' hints to manage data more accurately. The marked data objects are allocated to DRAM memory frames promptly, and they stick to the DRAM pages before the write reference counter times out. Thus the user marked data have a higher priority in competition for DRAM memory resource but won't retain the DRAM pages once they become cold in the new phase of application.
- **AllDRAM:** Here we implement a baseline policy in which there is sufficient fast memory to serve all pages in the system and no page movement is required. This serves as a nominal, best-case but impractical memory performance design.

3.3.2 Results

3.3.2.1 Energy Saving

Energy budgets are high restricted for mobile computing or embedded systems, making energy budgets a primary factor to consider in system design. Emerging NVM has negligible background power compared to the traditional DRAM technology, rendering an enormous advantage in energy saving versus DRAM. Here we examine the energy consumption of our proposed HMMU-UserHints scheme under the benchmark applications as compared against prior work [62] and static schemes versus an AllDRAM baseline. We normalize the energy consumption of our policies to

that of the AllDRAM configuration and present them in the figure 3.2. The figure shows that our

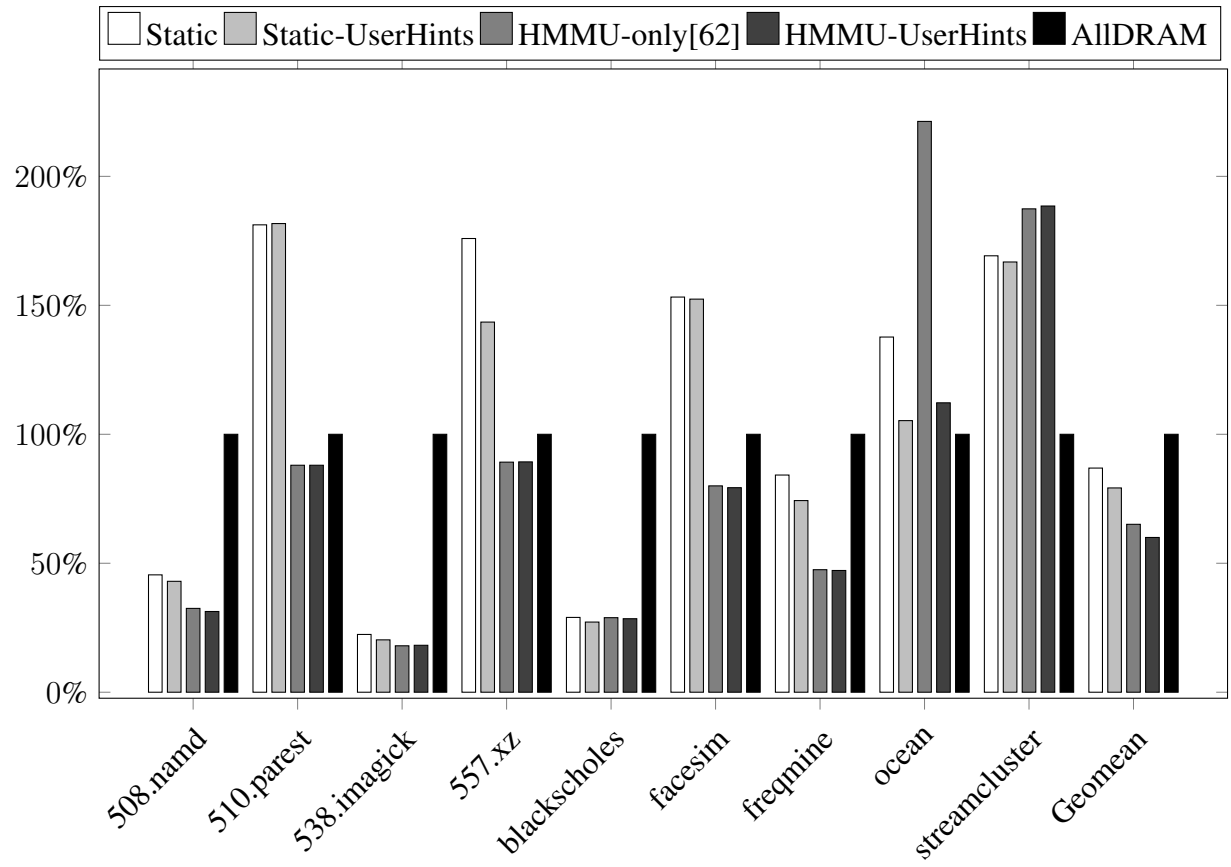


Figure 3.2: Energy Consumption Comparison

proposed hardware/software cooperative solution has the lowest energy consumption, $\sim$40% less than the allDRAM configuration. Our scheme outperforms the previously proposed HMMU [62] without user-hints, which achieves a $\sim$35% energy saving. Even without the underlying dynamic HMMU policies, the static allocation still significantly improves after adopting the user hints, as the Static-UserHints beats the baseline static allocation by 10% percent in terms of energy saving. This evidence validates the value of user hints in saving energy and the lifetime of NVM device.

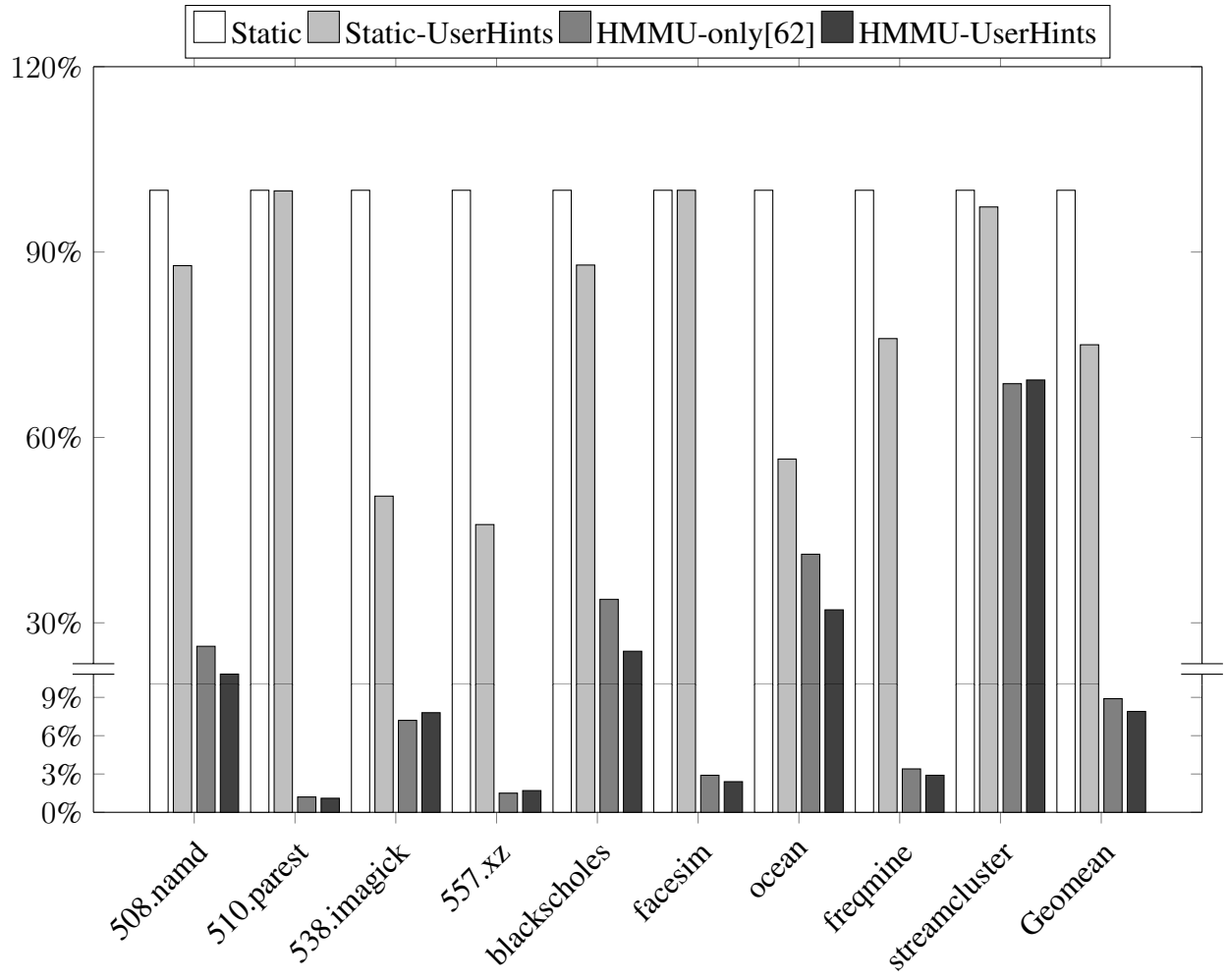


Figure 3.3: Writes to NVM

3.3.2.2 Writes Reduction and NVM Lifetime Saving

Writes in NVM technologies have 3x the latency and dissipate 8x the energy versus reads [7]. Further, NVM are susceptible to write cycle induced wearout. Figure 3.3 shows the number of writes to the NVM for the four tested configurations, with all data normalized against the baseline, random static allocation. Here we count not only the writes accesses generated by the CPU, but also the writes induced by the data migration triggered inside the HMMU. The figure shows that HMMU-UserHints policy outperforms the HMMU-only design by a margin of 14%. The gap expands to 92% when compared to the static allocation. The vast majority of writes were absorbed

in DRAM after the pages migrated from NVM where those writes accesses should have landed.

Figure 3.5shows a breakdown of the percentage of write accesses of DRAM or NVM, generated by Direct references or Data Migration respectively. The bar heights represent the total amount of writes normalized against the HMMU-only policy. Writes to NVM decreased in 7 out of the 9 applications, with the largest improvement in blackscholes. This application sees a 25% drop in overall NVM writes. Further, the part induced by data migration was reduced remarkably 28% compared to the HMMU-only design without user hints. The total number of writes including DRAM also diminished with these benchmarks, as the result of less data migration.

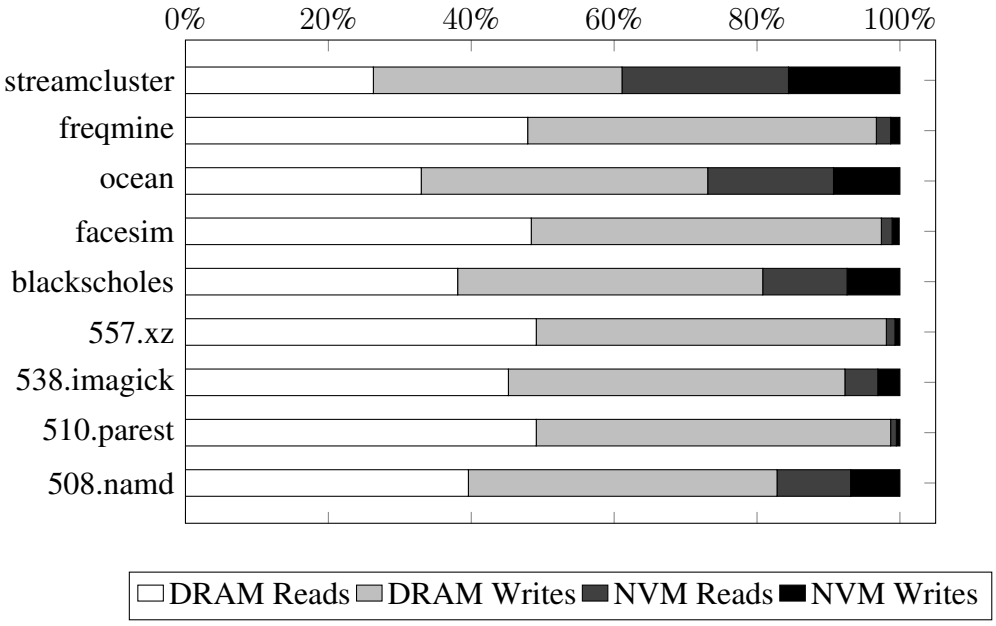


Figure 3.4: Memory Access Breakdown

3.3.2.3 Runtime Performance

Figure 3.6 shows the speedup attained by the different designs for the given benchmarks. We normalized all the data to the runtime of the AllDRAM configuration, which is the upper bound for all policies. The geometric mean performance across all applications is (in ascending order): Static (35.1%), Static-UserHints (39.9%), HMMU-only(77.6%), HMMU-UserHints(82.1%). This

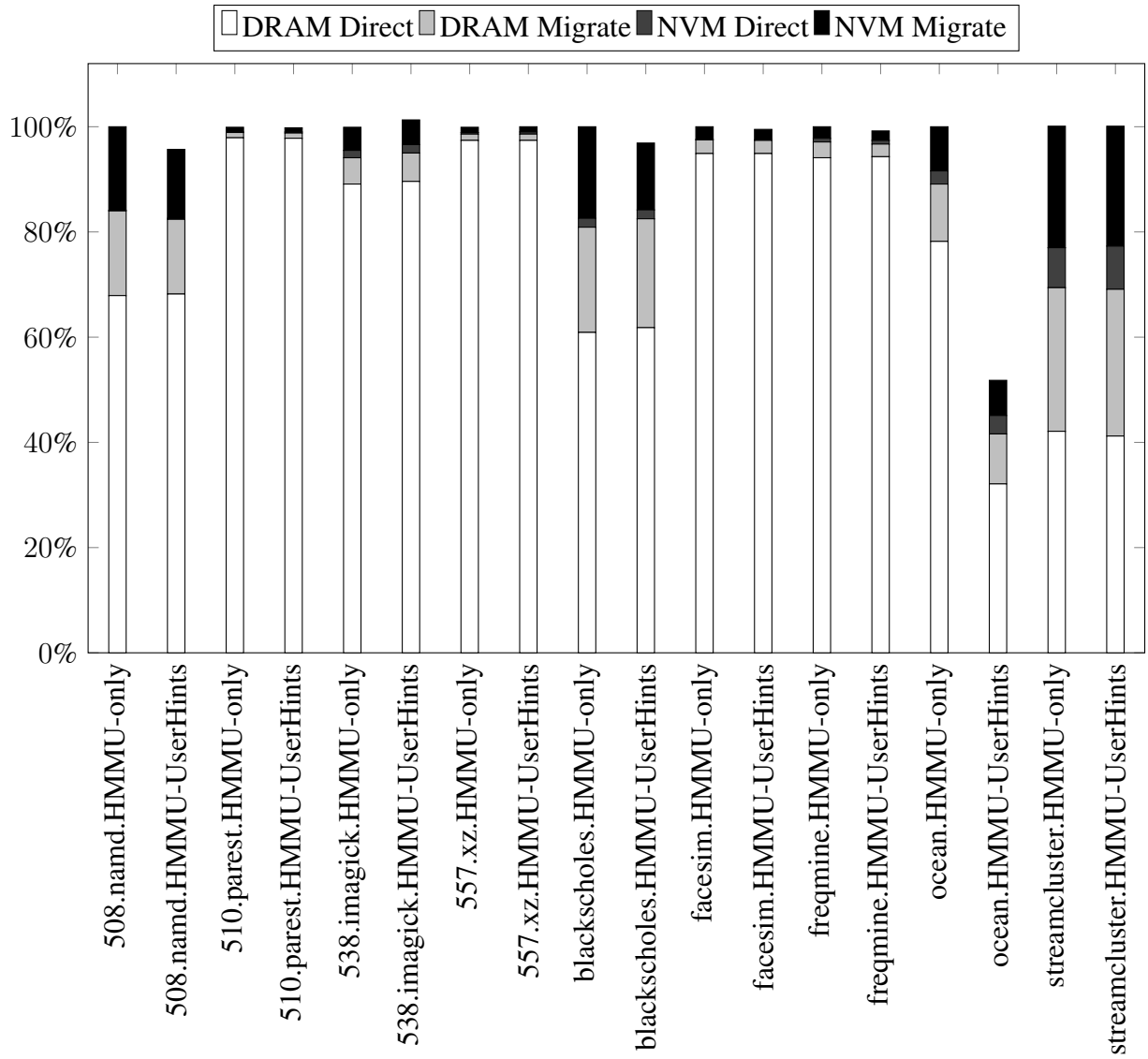


Figure 3.5: Writes Accesses BreakDown

result meets our expectation that the HMMU-UserHints has the best overall performance since it employs both the HMMU and knowledge from the programmers. Moreover, it achieves 97% of AllDRAM, if the two outliers, ocean and streamcluster, are excluded. We explore further details of memory accesses in Fig.3.4 to show the source of performance gain. Although DRAM only comprises 1/8 of the total memory capacity, we see it captures the vast majority of read requests. This explains why our hardware/software cooperative solution generally approximates AllDRAM performance. Compared to the HMMU-only design, our proposed solution was able to absorb

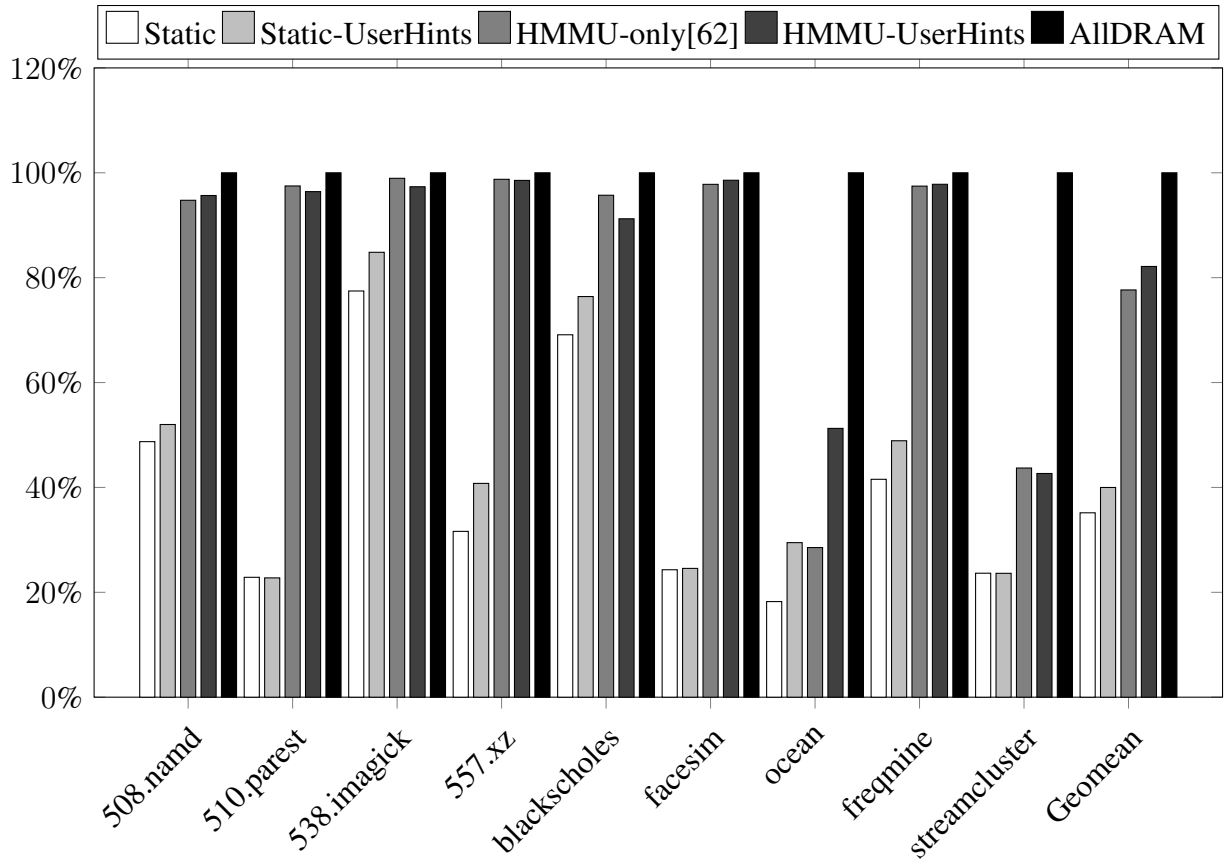


Figure 3.6: SPEC 2017 PARSEC Performance Speedup

more writes in the DRAM. The blackscholes application, for an instance, saw a 14% higher hit rate in the DRAM for those write requests sent from the CPU.

3.3.3 Specific Benchmark Analysis and Discussion

Ocean and streamcluster are the two benchmarks which don't benefit much from our policies in terms of energy saving and runtime performance. Ocean's data structures are designed in a way that prevents contiguous memory allocation, leading to poor locality. Its miss rate in cache is the second highest across all parsec benchmark applications. Moreover, this workload is highly sensitive to cache capacity, and the miss rate is two orders of magnitude higher for small caches. Streamcluster is a data-mining application. It is reported to have the the most core-to-bus data transfers [65]. Consequently streamcluster is very sensitive to memory bandwidth and access

latency. Therefore the performance is expected to degrade as we replace 7/8 of the DRAM resource in system with NVM which has significant higher latency. Both applications have poor data locality and their performance decline drastically with small CPU cache capacity. Thus we consider the performance loss is mainly attributed to other bottleneck parts of our testing system, other than the memory management itself. The HMMU-UserHints obtained the largest edge over HMMU-only policy in regard to NVM writes reduction with two benchmarks, 508.namd(18%) and ocean(22%). We inspected the number of marked pages that still remained in the DRAM after each application completed, and found that they have the highest percentage: 508.namd(46%) and ocean(69%). This metric to some extent shows how closely the user's hints match the actual accessed pattern of the marked data objects throughout the application. As we mentioned in Section 3.2.5, the HMMU still makes independent decision on data management even after accepting the user hints. Thus when the marked data objects went cold after application switching to new phase, they won't retain the precious DRAM resource. We infer that the benefit of having user hints were maximized with those data objects that keeps receiving frequent write references for the whole duration.

3.4 Summary

Emerging non-volatile memory (NVM) technologies provide higher capacity and less static power consumption than traditional DRAM. These features are promising to address the dilemma of the memory system on mobile computing/embedded system: new applications have ever-increasing memory footprint, while the limited battery life prohibits DRAM from scaling. NVM, however, tends to have longer access latency and write endurance issues. While prior work proposed purely hardware based schemes to manage this hybrid memory, we find that hardware resource limits impact the ability to detect long-term memory access patterns. Here we proposed a hardware/-software co-operative solution that incorporates users' hints of data properties. We customized the memkind allocator [63] to allow users define the target memory device. We also codesigned a HMMU so that the information collected from software stack could effectively collaborate with the existing hardware policies. We tested our scheme with benchmarks selected from SPEC 2017 and PARSEC suites. Experimental results show that our solution consumes remarkably 40% less

energy than an untenable all DRAM configuration, with a margin of 9% over the baseline HMMU solution. We also show our proposed scheme successfully reduced 14% writes to the NVM versus prior work. Our hardware/software cooperative solution managed to perform within 16% of the all DRAM configuration with only 1/8 DRAM capacity. This performance is 6% better than prior work.

4. INTERCONNECTION NETWORK WITH PHOTONIC LINKS

This chapter shows my exploration in the photonic-link interconnection network design for HPC/Data center. First I present the basics of photonic links and interconnection network, including the primary formula and algorithms used to optimize parameters in network design. Then I give a short introduction about Supersim, the interconnection network simulator that I used in the studies, along with several functions that I developed for it. In the third section, I illustrate the composite switch design, followed by the simulation results analysis in the end. Latency and power consumption has become major concerns of on-chip interconnects, especially in long-distance communication. These concerns can be alleviated in photonic interconnect because the latency and energy efficiency is less dependent on distance. Another advantage of photonic interconnects is much higher bandwidth and bandwidth density, which is an increasing requirement for larger number of cores per chip. In addition, photonic signal suffers less propagation loss than electrical signal. While the unique characteristics of photonics can help to overcome certain drawbacks of electrical signaling, they also inevitably impact the existing network design.

4.1 Photonic Interconnect Basics

Generally speaking, photonic links require three components: transmitter, transmission medium, and receiver. Transmitter converts electrical signal into photonic signal, which travels via transmission medium to destination node where they are restored to electrical signal by receiver.

4.1.1 Transmitter

There are two ways of light modulation: direct modulation and indirect modulation. The latter one is usually chosen for device that demands high area-efficiency, as it is comprised of micro-ring resonators and external laser source. Another reason for using external laser source is that silicon material is indirect-band and thus less power efficient to provide stimulated emission of photons. Lasers with WDM spectrum can carry parallel bits in a single transmission medium, but have higher cost and power consumption.

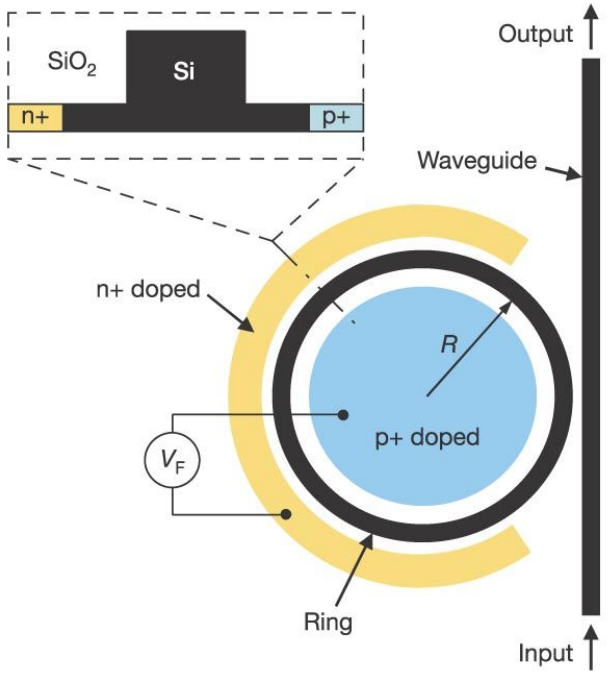


Figure 4.1: Microring Modulation "Reprinted from [4]"

Most laser sources nowadays are placed off-chip due to power limits, thus microring resonators become the most critical on-chip component of the transmitter. Microring resonators can be tuned to respond to specific wavelength and work as a filter. The wavelengths that a microring responds to repeat at a certain interval, which is called as Free Spectral Range (FSR).

However, microring is not a perfect filter and the pass band it creates matches a non-linear function with roll-offs. FSR also aggravates the undesirable cross-talk issues[66]. As the Shannon-Hartley theorem asserts, the channel capacity is defined by the bandwidth and its SNR. Therefore a higher and wider pass band with sharper roll-off is needed to improve channel capacity. The quality factor Q is measured by the 3db bandwidth of the microring resonator. The target Q limits microring size[67], which is a critical factor in on-chip photonic network design. We will illustrate it in more details in following sections.

There are two ways to tune the resonance frequency of the microring. The resonance frequency of a passive microring is determined during the fabrication process and cannot be changed after-

wards. The active microring, on the oppsite, can shift its resonance frequency by carrier injection or carrier depletion, therefore only active microring can be used for modulation. While carrier injection requires higher voltage, it provides a broader range to shift frequency and thus is more widely used. However, carrier injection also induces an adversary side effect: the attenuation of signal. Additionally, it is also worth mentioning that the resonance frequency of microring is sensitive to temperature.

There are two modulation schemes: modulating zeroes or modulating ones. These schemes are denoted by which logic value needs to actively created. In the modulating zeroes scheme, the incoming waveguide is also the outgoing waveguide, and we need to actively remove a certain wavelength to get a zero, while the unaffected wavelengths go through as logic 1. The modulating ones scheme is applied when incoming waveguide is separate from the outgoing one: we need to bend the wavelength onto the outgoing waveguide to create ones, while the rest continue uninterruptedly as zeros.

4.1.2 Transmission Medium

Although photons could be carried via a large variety of materials, here we only consider the silicon waveguide for better integration with other parts of the chip. Among the different configurations of silicon waveguides, channel and ridge waveguides are chosen because they provide single-mode propagation and thus are free from modal dispersion.

4.1.3 Receiver

The receiver is composed of microring, photo-detector and an amplifier. The energy consumption is the primary concern in the amplifier design, which can be divided to static power and active power (present when the amplifier is turned on). However, some research groups[68] also proposed a "receiver-less" design without amplification, on the premise that the capacitance of photo-detectors is low enough.

The photo-detector converts optical signal back to electrical signal. It requires certain amount of optical energy to generate required voltage of outcome electrical signal. The energy required

at photodector is proportional to the energy demanded by the light source and therefore it's a key factor to the energy efficiency of entire photonic link.

4.1.4 Photonic link

As mentioned in the previous paragraph, the size of microring is a significant factor for on-chip photonic network design. There is a limit on how small the microring can get, in order to guarantee Q factor of certain standard. Since the microrings cannot scale along with the electrical feature size, an imbalance appears between the power consumption of photonic transmitters and the supporting electronic components. We have to make trade-offs between multiple elements including power consumption, number of parallel bits, and signaling rate.

Suppose the external laser source provides a set of n wavelengths, and the transmitter electrical interface is m bits wide. These m-bit flit needs to go through a $m\colon n$ serializer before being fed to electrical drivers. The electrical driver will then convert these signals into drive currents for the microrings. At the receiver end, the microrings first extract the wavelengths from incoming waveguide. Then the photo-detectors will convert the optical signals back to electrical signals, and feed them to a $n\colon m$ deserializer. If amplification is needed, there will be a TIA stage between the output of photodetector and the input of deserializer.

There are three major parameters to consider in a interconnect design: the amount of bandwidth, the area and energy cost of that width.

4.1.4.1 Photonic Power Requirement

The required power of laser source is $P_PD \times 10^{\frac{A}{10}}$ where P_PD is the power required at photodetector, and A stands for the path attenuation. The minimum $P_PD = E_L * f_{mod}$,where E_L represents the energy needed for photodetector to switch, and f_{mod} is the signaling frequency. The microrings also attenuate the signals. For active microrings, the attenuation deteriorates as the current injection increases. Passive microrings attenuates the signal when routing them from one waveguide to another. Ahn *et al.* [69] and Nitta *et al.* [70] pointed that photo-detector also attenuates signal. The power also plays a part in deciding the bandwidth. Bandwidth is the product of

phit width and frequency. Thus we have to made a choice: a narrower/faster link or a wider/slower link? While a narrower link needs less microrings, a higher frequency requires a wider 3db communication bandwidth of the microring. More carrier injection is needed to shift the cut-off point further down the frequency spectrum and thus leads to signal deterioration.

4.1.4.2 Electronic Power Requirement

To deal with the variance of resonance frequency induced during microring fabrication or environment change, additional power is needed to maintain the correct frequency, which is called the trimming power. Nitta *et al.* [71] showed that the trimming power has a non-linear correlation to the number of microrings in the system-level. Other factors such as crosstalk and the positive feedback of current injection makes the analysis of trimming power even more complicated. However, trimming power can be considered as static overhead power since it's independent on frequency or phit width.

The power of modulation is dependent on microring capacitance and the switching frequency. The serializer and deserializer power consumption increases as the signal link become narrower and faster.

The power to support local transport within a node is another new factor introduced by using photonic link. Suppose the size of microrings is 3μm while the electrical feature size is 16 nm, then the size difference is as large as four orders of magnitude. Figure 4.2 gives an example to illustrate such the difference.

The matrix shown in upper left corner in fig 3 is the relative size of 128-bit buffer (4×32) made of D flip-flops. Why don't we lay out the microrings in grid instead of a single line? The reason is that bending waveguids significantly increase signal attenuation (0.1db per 90 degrees).Joshi *et al.* [72] Pan *et al.* [73] also assert that using microrings grid will increase design complexity. Vantrease *et al.* [74] showed that the number of waveguides have a great impact on final layout.

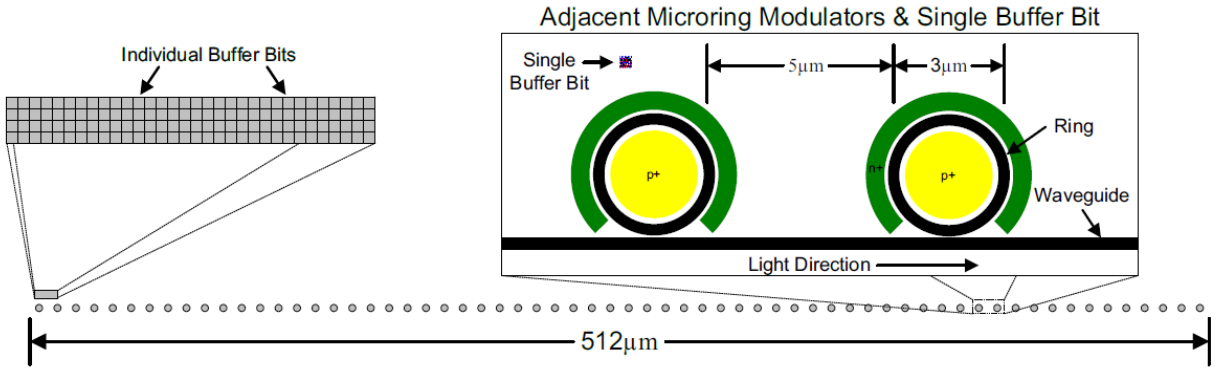


Figure 4.2: Relative sizes of electrical and photonic components "Reprinted from [5]"

4.1.5 On-chip Photonic Network

Unlike the electrical signals, that photons cannot be easily buffered or stored. This feature brings several challenges to the topology design, one of which is the arbitration. An intuitive solution is to convert the optical signal to electrical signal before entering the arbitration stage and restore it back to optical signals afterwards. This solution significantly increases both design complexity and latency. Vantrease *et al.* [74] proposed an arbitration scheme that operates in optical domain for the Corona system. Shacham and Bergman [75] presented another solution that uses electrical signaling for path establishment but offload the actual data packets onto optical signal. Nitta *et al.* [76] designed a directly connected network that totally avoided arbitration.

Electronics and photons has different advantages and disadvantages, rendering electrical/optical co-design is a reasonable choice for on-chip network design. A possible implementation is a two-level hierarchy design, i.e, processors within short distance are connected with electrical links while optical links covers long range communication among processor groups globally. Another solution is to implement electronic on-chip network with optical express channel. The latency of photonic network can be approximated by the following equation:

$$Latency = T_{send} + (d+1)T_{prop} + (T_r + T_a + T_s)d + \frac{\text{Pktsize}}{BW} + T_{rec}$$

The T_{send}, T_r, T_a, T_s and T_{rec} represents the latency incurred at sender, routing, arbitration, switching and receiver, receptively. d is the number of pops between source and destination node, and T_{prop} is the propagation delay. In regard to photon's inability to be buffered, there should be less hops to reduce latency. In order to better utilize the high bandwidth provided by photonic link, the packet size should be larger. Techniques such as packet aggregation may be used.

4.2 Photonics NoC

Small-size NoCs typically use bus to interconnect cores, which does not scale well as the number of cores increases, due to wire latency and contention. Mesh is another commonly used topology. Despite that it avoids long wire, mesh topology still suffers from contention and intra-node latency among embedded cores. In addition, mesh and torus typologies both require complex routers and buffers, ending up with increased energy and area consumption [77]. Therefore we will discuss some state-of-art typologies which fit better with large-scale systems, such as flattened butterfly [78], composite stitches, etc.

Multiprogramming is essential to exploit the potential parallelism of many-core systems. However, it also brings in new challenges to Noc System architecture. Multiprogramming incurs tremendous global messages due to process control operations and global data movement such as reduction, replication, permutation, segmented scan and barrier synchronization. Besides these global messages generated by applications, the shared cache system and cache coherence protocols also created significant amount of multicast messages across the network.

In the traditional NoCs, multicasting was achieved by a naive way that duplicates unicast messages for every single core. These unnecessary message replications might lead to network congestion and also waste of energy. Such problems exacerbates as the network size scales up. Recent work proposed tree based multicast algorithm, which requires additional hardware complexity to storage global routing information. Optical interconnections based on ring resonators is a natural fit for multicasting problem, as the signals propagated on the waveguide could be listened by multiple receivers simultaneously [79]. In spite of the physical advantages, several problems still remain in using photonic links for multicasting. The optical-electrical signal conversion in-

terface is not efficient, thus little photonics energy were left in the waveguide after the tune ring filtering[79]. Some works attempted to solve this problem using partially de-tuned ring filters, which retained significant portion of energy in the waveguide. Other works took advantage of the WDM (Wavelength Division Multiplexing) technique, via which information could disperse across multiple wavelength channels. The outcome is equivalent to a high-bandwidth multicasting channel.

Aside from providing natural mechanism for multicast to help cache coherence, the photonics NoC can also help to eliminate complicated cache coherence protocols due to races.

It is hard to ensure the memory consistency in the cache coherence protocols, especially in the topology like mesh which cannot naturally provide a total order in the network. In the work done by Vantrease, they proposed a photonics NoC which can simplify the complicated cache coherence protocol using mutex to serialize a race path. The mutex is implemented by taking advantage of the ultra-low latency of the photonics network.[80]

4.3 SuperSim Simulatror

When it comes to interconnection architecture design, we need to consider a wide scope of parameters and options. This list includes but not limited to: number of routers/nodes, bus protocol, fabrics, routing algorithms, network topology, router micro-architecture, arbitration mechanism. Moreover, these design choices are usually correlated to each other, hence it is impossible to investigate each decision point manually. To find the approximately optimal solution we need a simulator that could explore the design space effectively. Supersim [27] is a simulator project initiated by Nic McDonald at the Hewlett Packard Labs. I developed several new features and functions based on that framework, and conducted studies on two million-node scale network architectures. Compared to the existing network simulators, supersim has these following advantages:

1. Modularity for extension and flexibility

2. Significant improvement in simulation time due to event driven simulation

3. Realistic endpoint modeling

4. Capability to simulate large scale network

5. Independent configuration files

A major cause of the high overheads in many simulators is that they model the system's state cycle by cycle. Even when the system is in the idle state, the simulation is still performing computation. Based upon this observation, the supersim is designed as an event-driven simulator.
Events generated by all valid system components are inserted to the global event queue in chronological order. The simulator skips all time slots that bear no events and fast-forward to the next event in the queue. Thus we save all the unnecessary computation of the system states in idle cycles. In supersim, time is modeled as abstract units and the clock cycles are handled by a dedicated utility function. Each cycle is divided to two stages: "processing" and "interaction". In the processing stage every component will process the self-created events while in the "interaction" stage it either communicate with other components or create new events.

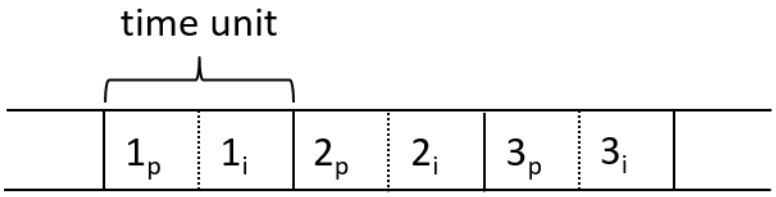


Figure 4.3: Clock cycles simulation

Supersim has a modularized framework, which we could interpolate a variety of components for each module. For instance, the terminal module could be instantiated as a simple message blaster for stress test, or a typical memory/process node for regular simulation. We could also freely manipulate the connections between terminals. A few example network typologies are illustrated as below:

Furthermore, we could also alter the micro-architectures of the switchlets:

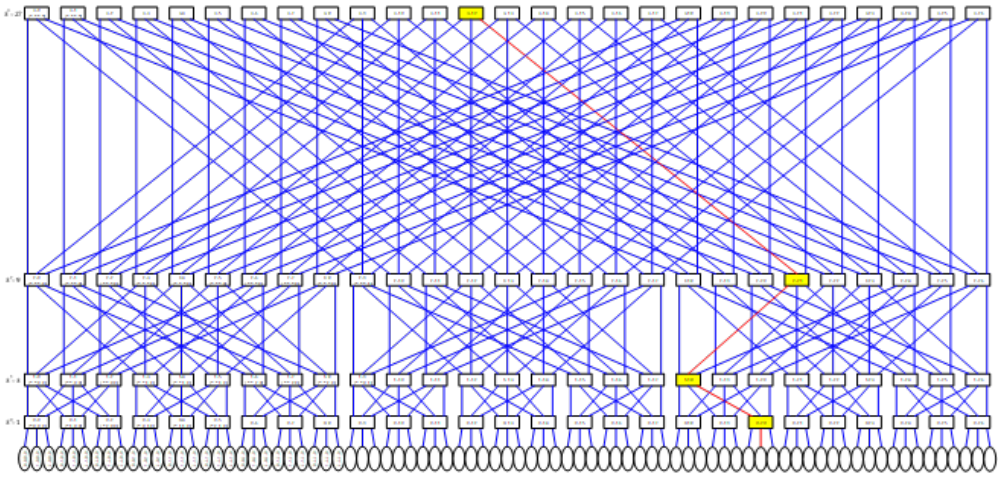
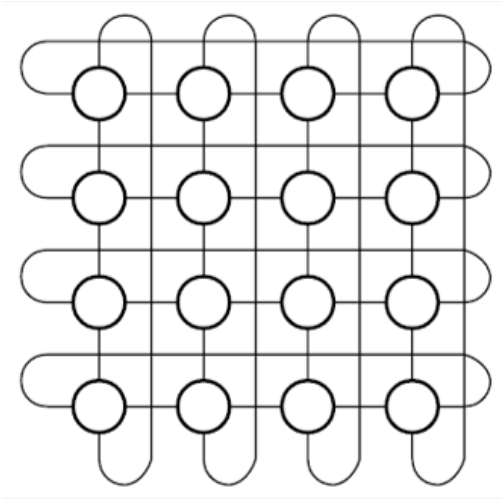
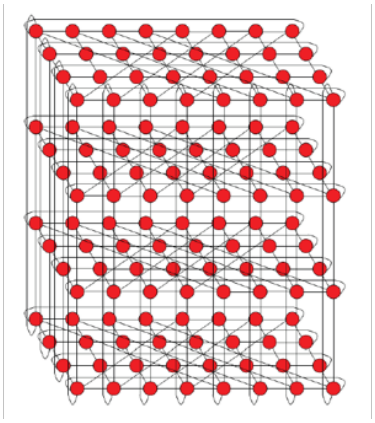


Figure 4.4: Example Topology 1: Butterfly

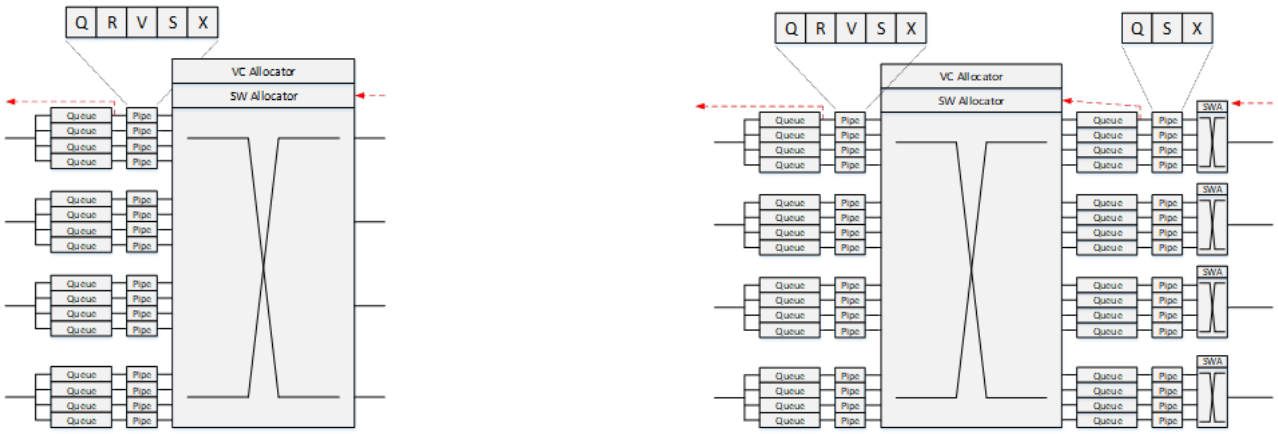


(a) Example Topology 2: 2D Torus



(b) Example Topology 3: 3D Torus

Figure 4.5: Sample Topologies



(a) Example Router Architecture 1: Router with input buffers only

(b) Example Router Architecture 2: Router with both input and output buffers

Figure 4.6: Sample Router Architectures

4.4 Composite Switch and Corona System Simulation

The design of a new network architecture could be divided into these steps:

1. Decide the underlying device properties, such as the physical channel bandwidth, distance (delay) of hops, message formats, etc.

2. Obtain the proper radix of router by $k \log^2 k = \frac{Bt_r \log N}{L}$, where B is the total channel bandwidth, t_r is the delay of each hop, N is the number of nodes in network and L is the length of a packet.

3. Router micro-architecture design, including the pipeline stages, number of virtual channels, resource allocation/de-allocation schemes.

4. Emulate all the possible combinations of number of radix, nodes, and topology that meet the target performance in terms of network scale, throughput, latency guarantee, etc.

In the first network architecture design, we found out the radix of a single switchlet was constrained by two factors:

- The crossbar and arbitrator cannot scale well along with the number of input/output connections.
- The complexity of wiring, area and power efficiency will deteriorate when the number of connections grows beyond certain point.

Concluded from these facts, we chose to build the network with composite switches, which are a bundle of simpler switchlets. Unfortunately the original supersim did not support network within network. Therefore I created a new method to simulate this new type of network:

- Generate the heat map of traffic traversing one composite switch with the data collected from the global network simulation.
- Recreate the traffic pattern obtained from the heat map, and study the performance of composite switch under this traffic.

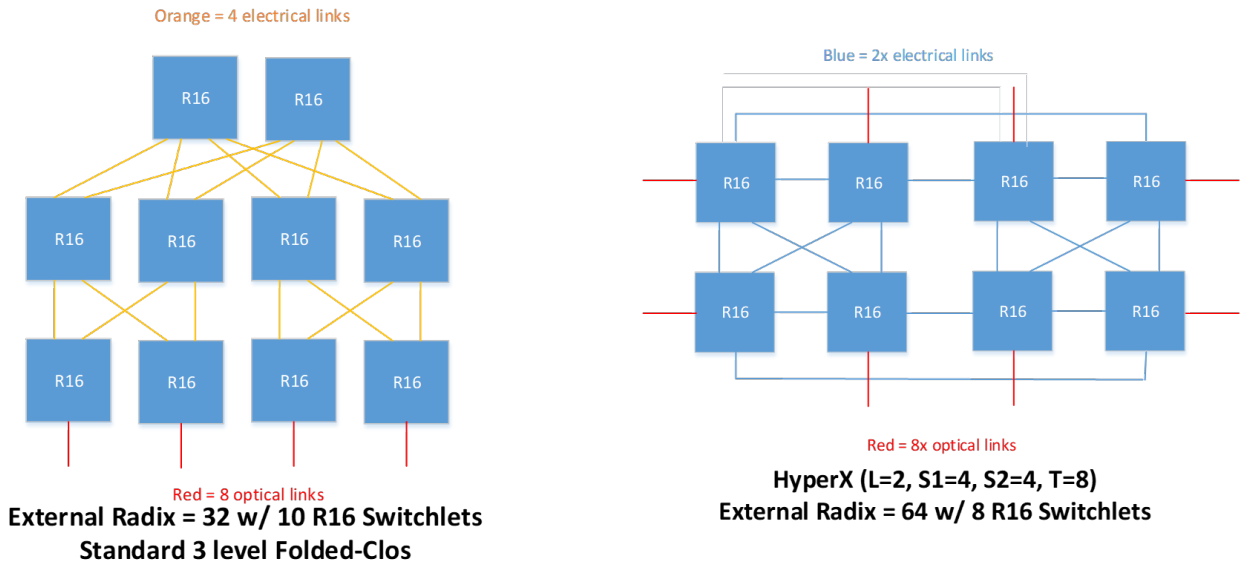


Figure 4.7: Example Composite Switch Architectures: folded-clos and HyperX[6]

Corona [81] is a nano-photonic interconnection network invented by Vantrease *et al.*. In the

second project, I implemented the Corona model in supersim and evaluate its performance. Basic facts about the corona architecture:

- 256 multi-threaded cores organized in 64 four-core clusters.
- Each core has a private L1 cache, L2 cache shared within cluster.
- A fully-connected 64*64 crossbar, implemented by 64 channels with MWSR (Multiple Write Single Read)
- Each channel bandwidth is 256 bit (4 bundled wave-guides * 64 wavelengths / wave-guide)
- System clock 5Ghz, modulation works at Double data rate, 10Ghz
- Crossbar bandwidth is 256 bit * 10Ghz * 64 =20.48 TB/s
- Token Arbitration

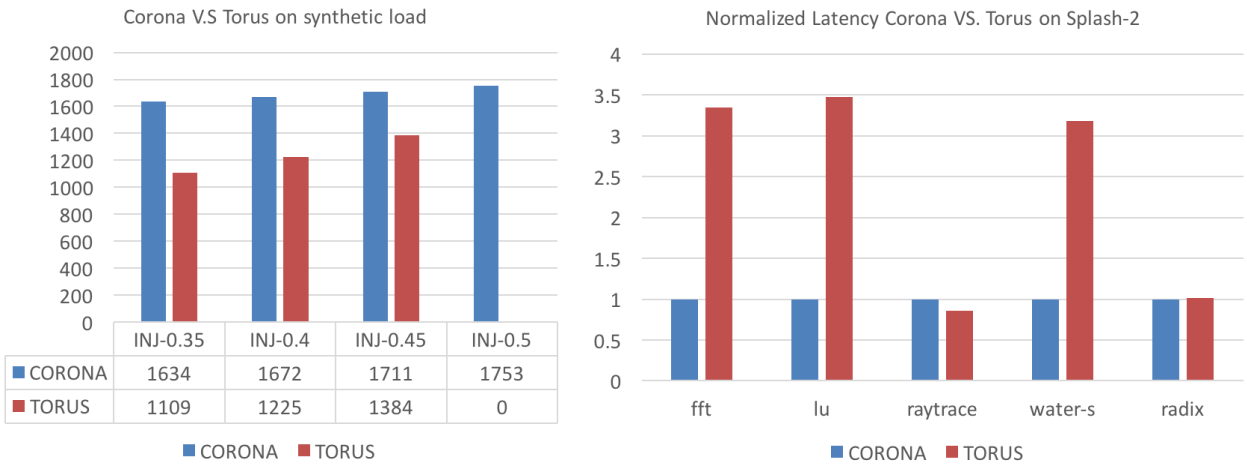


Figure 4.8: Corona V.S Torus

I compared the performance of Corona against torus interconnection network under synthetic workloads and splash-2 benchmark suite. Under the synthetic workloads, Corona showed higher

latency than torus network if we set the frequency of both as 10Ghz, which is impractical for electrical link. When I set the frequency of torus to 3.3Ghz, a commonly used value in industry, Corona exhibits a huge improvement of latency in all benchmark applications except the Raytrace. Raytrace is a 3D vision rendering application, which has a lot of discrete short messages targeted at different destinations. Such property stress the arbitration of Corona network, while the smaller size of messages could not fully exploit the benefit of larger bandwidth. My notions of corona network after analysing these results are:

- Token arbitration itself is not highly efficient, compared to traditional arbitration method (deadline, age). Even in no-contention scenario, the sender might need to wait a long time.
- The advantage of token based arbitration, is simplicity and parallelism, which fits the high frequency nano-photonic link better.
- The design principle of corona network takes into the difficulty of realization. That is why it choose the token arbitration, and serpentine optical crossbar.

4.5 Summary

In this chapter, we first pointed out that photonic link, as a new signal communication mechanism, is promising to address the bandwidth and power consumption issues for scalable system. Then we introduced the basics of photonic links in physical layer, data link layer and the interconnect network layer. In the next section, we introduced Supersim, a highly efficient network simulator that we used to explore the vast design space. I listed the general procedure and the formulas to decide on the basic parameters of an interconnect network. We also illustrated a few example network designs especially the composite switch design, and the two-fold simulation method that I invented. Then we presented the simulation result and analytical analysis of the Coronus architecture, in comparison to the traditional network with electrical links.

5. CONCLUSION

Big data drives two trends in the IT industry: memory intensive workloads demands more from the memory system, while the larger HPC/Data Center requires the interconnection network for higher bandwidth and radixes/connections. On the other hand, the escalation of memory system and interconnection network are both constrained by the power budget. Fortunately several emerging technologies shed light on paths to solve these problems. In this dissertation, I showed the quest to utilize non-volatile memory (NVM) for the mobile computing hybrid memory system, and to build high-radix interconnection network with photonic links.

A wide spectrum of non-volatile memory (NVM) technologies are emerging, including phase-change memories (PCM), memristor, and 3D XPoint. These technologies look particularly appealing for inclusion in the mobile computing memory hierarchy. While NVM provides higher capacity and less static power consumption, than traditional DRAM, its access latency and write costs remain problematic. Integration of these new memory technologies in the mobile memory hierarchy requires a fundamental re-architecting of traditional system designs. We first presented the Hardware-based Memory Management Unit(HMMU) that addresses both types of memory in a flat space address space. We also designed a set of data placement and data migration policies within this memory manager, such that we may exploit the advantages of each memory technology. While the page move policy provided good performance, adding a sub-page-block caching policy helps to reduce writes to NVM and save energy. On top of these two fundamental policies, we built an adaptive policy that intelligently chooses between them, according to the various phases of the running application. Experimental results show that our adaptive policy can significantly reduce power consumption by almost 40%. With only a small fraction of the system memory implemented in DRAM, the overall system performance comes within 12% of the full DRAM configuration, which is more than 2X the performance of random allocation of NVM and DRAM. By reducing the number of writes to NVM, our policy also helps to extend device lifetime. In the second chapter, we proposed a hardware/software co-operative solution that incorporates users'

hints of data properties. While the prior work proposed purely hardware based schemes to manage this hybrid memory, we find that hardware resource limits impact the ability to detect long-term memory access patterns. Here We customized the memkind allocator [63] to allow users define the target memory device. We also codesigned a HMMU so that the information collected from software stack could effectively collaborate with the existing hardware policies. We tested our scheme with benchmarks selected from SPEC 2017 and PARSEC suites. Experimental results show that our solution consumes remarkably 40% less energy than an untenable all DRAM configuration, with a margin of 9% over the baseline HMMU solution. We also show our proposed scheme successfully reduced 14% writes to the NVM versus prior work. Our hardware/software cooperative solution managed to perform within 16% of the all DRAM configuration with only 1/8 DRAM capacity. This performance is 6% better than prior work. In the third chapter, we first gave a literature review on the photnonic-link technology, and explained why it's a natural fit for the large scale interconnection network. The key features are that photonic links provides much higher bandwidth and is more robust against signal attenuation across long range transmission. We showed the primary parameters, formula and design flow of the interconnection network. We also brought in the idea of composite switch to cope with the physical property difference of electronic and photonic device. At last we conducted simulation on the Corona network to quantitatively show the how photonic links could help to improve the network performance in several metrics including throughput/latency, over the traditional network built of pure electronic links.

REFERENCES

[1] CISCO PUBLIC, "Cisco global cloud index 2016–2021," 2016. https://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.html.

[2] CISCO PUBLIC, "Cisco vni index 2015–2020," 2016. https://www.cisco.com/c/dam/m/en_in/innovation/enterprise/assets/mobile-white-paper-c11-520862.pdf.

[3] P. Kogge and J. Shalf, "Exascale computing trends: Adjusting to the "new normal"' for computer architecture," *Computing in Science & Engineering*, vol. 15, pp. 16–26, 11 2013.

[4] Q. Xu, B. Schmidt, S. Pradhan, and M. Lipson, "Micrometre-scale silicon electro-optic modulator," *Nature*, vol. 435, no. 7040, pp. 325–327, 2005.

[5] C. Nitta, M. Farrens, and V. Akella, "Evaluating the energy efficiency of microring resonator-based on-chip photonic interconnects," 2012.

[6] J. H. Ahn, N. Binkert, A. Davis, M. McLaren, and R. S. Schreiber, "Hyperx: topology, routing, and packaging of efficient large-scale networks," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, pp. 1–11, Nov 2009.

[7] A. Chen, "A review of emerging non-volatile memory (nvm) technologies and applications," *Solid-State Electronics*, vol. 125, pp. 25–38, 2016.

[8] S. Mittal and J. S. Vetter, "A survey of software techniques for using non-volatile memories for storage and main memory systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 5, pp. 1537–1550, 2016.

[9] J. J. Yang, D. B. Strukov, and D. R. Stewart, "Memristive devices for computing," *Nature Nanotechnology*, Dec 2012.

[10] SPEC, "SPEC CPU2017 Documentation," 2017. https://www.spec.org/cpu2017/Docs/.

[11] C. Bienia, *Benchmarking Modern Multiprocessors*. PhD thesis, Princeton University, January 2011.

[12] W. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003.

[13] R. Lucas, J. Ang, K. Bergman, S. Borkar, W. Carlson, L. Carrington, G. Chiu, R. Colwell, W. Dally, J. Dongarra, A. Geist, R. Haring, J. Hittinger, A. Hoisie, D. M. Klein, P. Kogge, R. Lethin, V. Sarkar, R. Schreiber, J. Shalf, T. Sterling, R. Stevens, J. Bashor, R. Brightwell, P. Coteus, E. Debenedictus, J. Hiller, K. H. Kim, H. Langston, R. M. Murphy, C. Webster, S. Wild, G. Grider, R. Ross, S. Leyffer, and J. Laros III, "Doe advanced scientific computing advisory subcommittee (ascac) report: Top ten exascale research challenges," 2 2014.

[14] INTEL CORPORATION, "Intel 750," 2015. https://ark.intel.com/products/86740/Intel-SSD-750-Series-400GB-12-Height-PCIe-3_0-20nm-MLC.

[15] Y. Kwon, H. Fingler, T. Hunt, S. Peter, E. Witchel, and T. Anderson, "Strata: A cross media file system," in *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, (New York, NY, USA), pp. 460–477, ACM, 2017.

[16] J. Zhang, M. Kwon, D. Gouk, S. Koh, C. Lee, M. Alian, M. Chun, M. T. Kandemir, N. S. Kim, J. Kim, and M. Jung, "Flashshare: Punching through server storage stack from kernel to firmware for ultra-low latency ssds," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, (Carlsbad, CA), pp. 477–492, USENIX Association, 2018.

[17] INTEL CORPORATION, "Intel optane technology," 2016. https://www.intel.com/content/www/us/en/architecture-and-technology/intel-optane-technology.html.

[18] K. Eshraghian, K.-R. Cho, O. Kavehei, S.-K. Kang, D. Abbott, and S.-M. S. Kang, "Memristor mos content addressable memory (mcam): Hybrid architecture for future high performance search engines," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 19, pp. 1407–1417, Aug 2011.

[19] S. Raoux, G. W. Burr, M. J. Breitwisch, C. T. Rettner, Y. . Chen, R. M. Shelby, M. Salinga, D. Krebs, S. . Chen, H. . Lung, and C. H. Lam, "Phase-change random access memory: A scalable technology," *IBM Journal of Research and Development*, vol. 52, pp. 465–479, July 2008.

[20] J. Choe, "Intel 3d xpoint memory die removed from intel optane pcm," 2017. `https://www.techinsights.com/blog/intel-3d-xpoint-memory-die-removed-intel-optanetm-pcm-phase-change-memory`.

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

[22] C. 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 *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 1–12, Dec 2014.

[23] A. Hassan, H. Vandierendonck, and D. S. Nikolopoulos, "Software-managed energy-efficient hybrid dram/nvm main memory," in *Proceedings of the 12th ACM International Conference on Computing Frontiers*, CF '15, (New York, NY, USA), pp. 23:1–23:8, ACM, 2015.

[24] V. Fedorov, J. Kim, M. Qin, P. V. Gratz, and A. L. N. Reddy, "Speculative paging for future nvm storage," in *Proceedings of the International Symposium on Memory Systems*, MEMSYS '17, (New York, NY, USA), pp. 399–410, ACM, 2017.

[25] Z. Wang, Z. Gu, and Z. Shao, "Optimizated allocation of data variables to pcm/dram-based hybrid main memory for real-time embedded systems," *IEEE Embedded Systems Letters*,

vol. 6, pp. 61–64, Sept 2014.

[26] E. D. Berger, B. G. Zorn, and K. S. McKinley, "Reconsidering custom memory allocation," *SIGPLAN Not.*, vol. 37, p. 1–12, Nov. 2002.

[27] N. McDonald, "Supersim repository." `https://github.com/HewlettPackard/supersim`, 2018.

[28] I. T. R. F. SEMICONDUCTORS, "Moremoore," 2015. `https://www.semiconductors.org/resources/2015-international-technology-roadmap-for-semiconductors-itrs/`.

[29] A. Shilov, "Pricing of intel's optane dc persistent memory modules," 2019. `https://www.anandtech.com/show/14180/pricing-of-intels-optane-dc-persistent-memory-modules-leaks`.

[30] N. Madan, L. Zhao, N. Muralimanohar, A. Udipi, R. Balasubramonian, R. Iyer, S. Makineni, and D. Newell, "Optimizing communication and capacity in a 3d stacked reconfigurable cache hierarchy," in *2009 IEEE 15th International Symposium on High Performance Computer Architecture*, pp. 262–274, Feb 2009.

[31] D. Jevdjic, G. H. Loh, C. Kaynak, and B. Falsafi, "Unison cache: A scalable and effective die-stacked dram cache," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 25–37, Dec 2014.

[32] 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 *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 235–246, Dec 2012.

[33] J. Sim, A. R. Alameldeen, Z. Chishti, C. Wilkerson, and H. Kim, "Transparent hardware management of stacked dram as part of memory," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 13–24, Dec 2014.

[34] J. Meza, J. Chang, H. Yoon, O. Mutlu, and P. Ranganathan, "Enabling efficient and scalable hybrid memories using fine-granularity dram cache management," *IEEE Computer Architecture Letters*, vol. 11, no. 2, pp. 61–64, 2012.

[35] C.-C. Huang and V. Nagarajan, "Atcache: Reducing dram cache latency via a small sram tag cache," in *Proceedings of the 23rd International Conference on Parallel Architectures and Compilation*, PACT '14, (New York, NY, USA), pp. 51–60, ACM, 2014.

[36] J. Meza, J. Chang, H. Yoon, O. Mutlu, and P. Ranganathan, "Enabling efficient and scalable hybrid memories using fine-granularity dram cache management," *IEEE Computer Architecture Letters*, vol. 11, no. 2, pp. 61–64, 2012.

[37] Y. Lee, J. Kim, H. Jang, H. Yang, J. Kim, J. Jeong, and J. W. Lee, "A fully associative, tagless dram cache," in *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, pp. 211–222, June 2015.

[38] X. Wu and A. L. N. Reddy, "Managing storage space in a flash and disk hybrid storage system," in *2009 IEEE International Symposium on Modeling, Analysis Simulation of Computer and Telecommunication Systems*, pp. 1–4, Sept 2009.

[39] H. Liu, Y. Chen, X. Liao, H. Jin, B. He, L. Zheng, and R. Guo, "Hardware/software cooperative caching for hybrid dram/nvm memory architectures," in *Proceedings of the International Conference on Supercomputing*, ICS '17, (New York, NY, USA), pp. 26:1–26:10, ACM, 2017.

[40] T. Johnson and D. Shasha, "2q: A low overhead high performance buffer management replacement algorithm," in *VLDB*, 1994.

[41] E. J. O'Neil, P. E. O'Neil, and G. Weikum, "The lru-k page replacement algorithm for database disk buffering," in *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, SIGMOD '93, (New York, NY, USA), pp. 297–306, ACM, 1993.

[42] J. Kim, S. H. Pugsley, P. V. Gratz, A. L. N. Reddy, C. Wilkerson, and Z. Chishti, "Path confidence based lookahead prefetching," in *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-49, (Piscataway, NJ, USA), pp. 60:1–60:12, IEEE Press, 2016.

[43] ChampSim, "Champsim," 2016. `https://github.com/ChampSim/ChampSim`.

[44] 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*, vol. 39, pp. 1–7, Aug. 2011.

[45] T. Nowatzki, J. Menon, C. Ho, and K. Sankaralingam, "Architectural simulators considered harmful," *IEEE Micro*, vol. 35, pp. 4–12, Nov 2015.

[46] M. L. Shay Gal-On, "Exploring coremark - a benchmark maximizing simplicity and efficacy," 2012. `https://www.eembc.org/techlit/articles/coremark-whitepaper.pdf`.

[47] EEMBC, "An eembc benchmark for android devices," 2015. `http://www.eembc.org/andebench`.

[48] J. Evans, "Jemalloc," 2016. `http://jemalloc.net/`.

[49] I. Micron Technology, "Calculating memory power for ddr4 sdram," tech. rep., 2017.

[50] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting phase change memory as a scalable dram alternative," in *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ISCA '09, (New York, NY, USA), pp. 2–13, ACM, 2009.

[51] A. Limaye and T. Adegbija, "A workload characterization of the spec cpu2017 benchmark suite," in *2018 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 149–158, 2018.

[52] J. C. Mogul, E. Argollo, M. Shah, and P. Faraboschi, "Operating system support for nvm+dram hybrid main memory," in *Proceedings of the 12th Conference on Hot Topics in Operating Systems*, HotOS'09, (USA), p. 14, USENIX Association, 2009.

[53] M. R. Meswani, G. H. Loh, S. Blagodurov, D. Roberts, J. Slice, and M. Ignatowski, "Toward efficient programmer-managed two-level memory hierarchies in exascale computers," in *Proceedings of the 1st International Workshop on Hardware-Software Co-Design for High Performance Computing*, Co-HPC '14, p. 9–16, IEEE Press, 2014.

[54] S. R. Dulloor, A. Roy, Z. Zhao, N. Sundaram, N. Satish, R. Sankaran, J. Jackson, and K. Schwan, "Data tiering in heterogeneous memory systems," in *Proceedings of the Eleventh European Conference on Computer Systems*, EuroSys '16, (New York, NY, USA), Association for Computing Machinery, 2016.

[55] Z. Yan, D. Lustig, D. Nellans, and A. Bhattacharjee, "Nimble page management for tiered memory systems," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, (3304024), pp. 331–345, ACM.

[56] K. Wu, Y. Huang, and D. Li, "Unimem: Runtime data managementon non-volatile memory-based heterogeneous main memory," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '17, (New York, NY, USA), Association for Computing Machinery, 2017.

[57] A. J. Peña and P. Balaji, "Toward the efficient use of multiple explicitly managed memory subsystems," in *2014 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 123–131, Sep. 2014.

[58] R. Salkhordeh and H. Asadi, "An operating system level data migration scheme in hybrid dram-nvm memory architecture," in *Proceedings of the 2016 Conference on Design,Automation and Test in Europe*, pp. 936–941, EDA Consortium.

[59] G. Dhiman, R. Ayoub, and T. Rosing, "Pdram: A hybrid pram and dram main memory system," in *Proceedings of the 46th Annual Design Automation Conference*, DAC '09, (New York, NY, USA), pp. 664–469, ACM, 2009.

[60] A. J. Uppal and M. R. Meswani, "Towards workload-aware page cache replacement policies for hybrid memories," in *Proceedings of the 2015 International Symposium on Memory Systems - MEMSYS '15*, pp. 206–219.

[61] S. Bock, B. R. Childers, R. Melhem, and D. Mossé, "Concurrent page migration for mobile systems with os-managed hybrid memory," in *Proceedings of the 11th ACM Conference on Computing Frontiers - CF '14*, pp. 1–10.

[62] F. Wen, M. Qin, P. V. Gratz, and A. L. N. Reddy, "Hardware memory management for future mobile hybrid memory systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 3627–3637, 2020.

[63] L. Anaczkowski, "User Extensible Heap Manager for Heterogeneous Memory Platforms and Mixed Memory Policies," 2019. `http://memkind.github.io`.

[64] N. Nethercote and J. Seward, "Valgrind: A framework for heavyweight dynamic binary instrumentation," in *Proceedings of the 28th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '07, (New York, NY, USA), p. 89–100, Association for Computing Machinery, 2007.

[65] M. Bhadauria, V. M. Weaver, and S. A. McKee, "Understanding parsec performance on contemporary cmps," in *2009 IEEE International Symposium on Workload Characterization (IISWC)*, pp. 98–107, 2009.

[66] K. Preston, N. Sherwood-Droz, J. Levy, and M. Lipson, "Performance guidelines for wdm interconnects based on silicon microring resonators," in *Lasers and Electro-Optics (CLEO), 2011 Conference on*, pp. 1–2, May 2011.

[67] Q. Xu, D. Fattal, and R. Beausoleil, "1.5 um-radius high-q silicon microring resonators," in *Lasers and Electro-Optics, 2008 and 2008 Conference on Quantum Electronics and Laser Science. CLEO/QELS 2008. Conference on*, pp. 1–2, May 2008.

[68] A. Bhatnagar, C. Debaes, R. Chen, N. C. Hellman, G. A. Keeler, D. Agarwal, H. Thienpont, and D. A. B. Miller, "Receiverless clocking of a cmos digital circuit using short optical pulses," in *The 15th Annual Meeting of the IEEE Lasers and Electro-Optics Society*, vol. 1, pp. 127–128 vol.1, 2002.

[69] J. Ahn, M. Fiorentino, R. Beausoleil, N. Binkert, A. Davis, D. Fattal, N. Jouppi, M. McLaren, C. Santori, R. Schreiber, S. Spillane, D. Vantrease, and Q. Xu, "Devices and architectures for photonic chip-scale integration," vol. 95, pp. 989–997, Springer-Verlag, 2009.

[70] C. Nitta, M. Farrens, and V. Akella, "Dcof-an arbitration free directly connected optical fabric," *Emerging and Selected Topics in Circuits and Systems, IEEE Journal on*, vol. 2, pp. 169–182, June 2012.

[71] C. Nitta, M. Farrens, and V. Akella, "Addressing system-level trimming issues in on-chip nanophotonic networks," in *High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on*, pp. 122–131, Feb 2011.

[72] A. Joshi, C. Batten, Y. Kwon, S. Beamer, I. Shamim, K. Asanovic, and V. Stojanovic, "Silicon-photonic clos networks for global on-chip communication," in *2009 3rd ACM/IEEE International Symposium on Networks-on-Chip*, pp. 124–133, 2009.

[73] Y. Pan, J. Kim, and G. Memik, "Flexishare: Channel sharing for an energy-efficient nanophotonic crossbar," in *HPCA - 16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture*, pp. 1–12, 2010.

[74] D. Vantrease, R. Schreiber, M. Monchiero, M. McLaren, N. Jouppi, M. Fiorentino, A. Davis, N. Binkert, R. Beausoleil, and J. Ahn, "Corona: System implications of emerging nanophotonic technology," in *Computer Architecture, 2008. ISCA '08. 35th International Symposium on*, pp. 153–164, June 2008.

[75] A. Shacham and K. Bergman, "Building ultralow-latency interconnection networks using photonic integration," *Micro, IEEE*, vol. 27, pp. 6–20, July 2007.

[76] C. Nitta, M. Farrens, and V. Akella, "Dcaf - a directly connected arbitration-free photonic crossbar for energy-efficient high performance computing," in *Parallel Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, pp. 1144–1155, May 2012.

[77] G. Kurian, J. E. Miller, J. Psota, J. Eastep, J. Liu, J. Michel, L. C. Kimerling, and A. Agarwal, "Atac: a 1000-core cache-coherent processor with on-chip optical network," in *Proceedings of the 19th international conference on Parallel architectures and compilation techniques*, pp. 477–488, ACM, 2010.

[78] J. Kim, J. Balfour, and W. Dally, "Flattened butterfly topology for on-chip networks," in *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*, pp. 172–182, 2007.

[79] C. Li, M. Browning, P. V. Gratz, and S. Palermo, "Energy-efficient optical broadcast for nanophotonic networks-on-chip," *Laser*, vol. 20, no. 20, p. 20, 2012.

[80] D. Vantrease, M. H. Lipasti, and N. Binkert, "Atomic coherence: Leveraging nanophotonics to build race-free cache coherence protocols," in *High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on*, pp. 132–143, IEEE, 2011.

[81] D. Vantrease, R. Schreiber, M. Monchiero, M. McLaren, N. P. Jouppi, M. Fiorentino, A. Davis, N. Binkert, R. G. Beausoleil, and J. H. Ahn, "Corona: System implications of emerging nanophotonic technology," *SIGARCH Comput. Archit. News*, vol. 36, pp. 153–164, June 2008.