Disintegrated Control for Energy-Efficient and Heterogeneous Memory Systems

Tae Jun Ham Bharath K.Chelepalli Neng Xue Benjamin C.Lee
Duke University
{taejun.ham, bc125, nx3, benjamin.c.lee}@duke.edu

Abstract

A mix of emerging technologies promises qualitatively new memory system capabilities. However, today's memory controllers and channels constrain heterogeneity. Today's integration of controllers on processor dies prevents systems from accommodating diverse, technology-specific protocols and schedulers in a modular manner; memory design decisions must be made during processor design. Moreover, today's channel architectures are not flexible enough to accommodate diverse demands for bandwidth and capacity.

To address these challenges, we present strategies for scalability and heterogeneity, which include (i) disintegrating memory controllers to support heterogeneous command protocols in a modular manner; (ii) adding buffers to ensure signal integrity; and (iii) organizing buffers hierarchically to reduce latency. We apply these strategies to architect a novel heterogeneous DRAM / PCM system. Finally, we present mechanisms for power-efficient data movement.

1. Introduction

Memory has long been a computing bottleneck [43], which has been exacerbated by the advent of multi-core processors. While the traditional memory wall concerned only latency and bandwidth, a new challenge emerges when provisioning requisite capacities for multi-cores at tractable power costs. These memory capacities are demanded by the rise of big data analytics (Hadoop), distributed memory caching (e.g., MemCached), and in-memory storage (e.g., RAMCloud). Unfortunately, with its power-intensive, high-speed interfaces and its periodic refresh, DRAM is ineffective at these scales. And without scaling, average memory capacity per computational task falls as the number of processors, cores, and threads increases.

Emerging technologies and command protocols complement DRAM in high-capacity memory systems. DRAM stores data by placing charge in a capacitor, which requires periodic refresh and high-performance DDR interfaces to accommodate high device bandwidth. In contrast, phase change memory (PCM) is a non-volatile technology that stores data by altering the phase of a chalcogenide. PCM is within competitive range of DRAM performance when architected with on-chip registers, which improve locality and filter writes [26, 34, 11, 46]. PCM dissipates little static power as a non-volatile, low-bandwidth technology, which uses a new power-efficient LPDDR2-N protocol[21].

With heterogeneity, an architecture can exploit PCM power efficiency and DRAM performance. However, heterogeneity faces several challenges. First, today's memory controllers are integrated with the processor, forcing architects to make memory design decisions during processor design. Such integration constrains heterogeneity as technologies prefer diverse command protocols. In addition, processor memory bandwidth is limited by pin count and low-bandwidth memories, like PCM, may waste valuable pins. Finally, data movement between heterogeneous technologies is expensive and intelligent protocols are needed for power efficiency. To address these challenges, we make the following contributions:

- **Partitioned Controllers.** For modular heterogeneity, we partition memory controller functionality between on-chip masters and off-chip slaves. Masters relay processor memory commands to slaves, which implement technology-specific protocols (§2).
- **Hierarchical Buffers.** For scalable capacity, we architect a many-rank system with buffers to ensure signal integrity. Because each buffer introduces a latency, we organize buffers hierarchically to reduce hop count (§3).
- **Hybrid PCM/DRAM.** For efficient heterogeneity, we provide capacity with many PCM ranks and provide performance with a few DRAM ranks. We manage DRAM as a fine-grained cache, proposing dynamic-granularity cache fills and read-only policies to reduce data movement and its associated power costs (§4).

We apply these strategies to DDR3 DRAM and LPDDR2-N PCM to demonstrate tractable power costs as main memory capacity scales to tens of ranks and hundreds of gigabytes (§5-6). For 32 ranks of main memory, a heterogeneous PCM/DRAM architecture incurs less than a 1.15× performance penalty and consumes only 0.2× the power of a homogeneous DRAM architecture. As system capacity increases, DRAM power costs increase rapidly while those for PCM are flat.

2. Architecting Heterogeneous Controllers

At present, high-performance processors integrate the memory controller onto the processor die. While integration reduces latency, it also brings unintended limitations in memory system heterogeneity. First, an integrated controller's size is limited by the processor die's area budget and accommodating different protocols and schedulers for different technologies could significantly increase area costs.

In addition, processor-memory bandwidth is limited by processor pin count. While conventional DRAM uses high-speed memory channels, other technologies operate channels at lower frequencies. Examples include LPDDR* for low-power DRAM and LPDDR*-N for phase change memory, which halve the peak channel bandwidth of DDR*. With integrated controllers, heterogeneous memory systems must

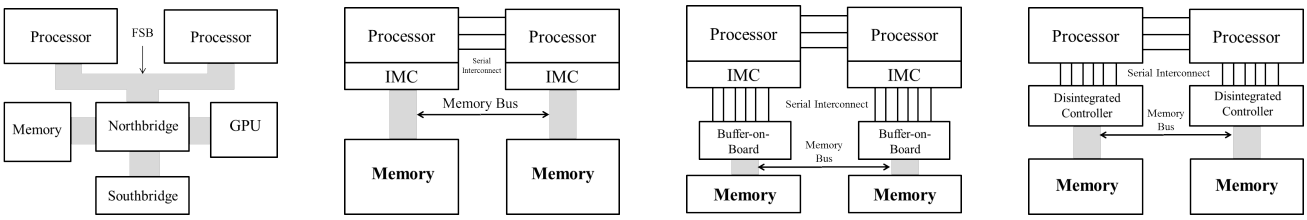

Figure 1: Memory controller architectures.

operate some of the scarce pins at lower data rates, reducing aggregate off-chip bandwidth for a given number of pins. Thus, heterogeneous technologies motivate a different approach and we propose disintegrated memory controllers.

2.1. Disintegrating Memory Controllers

By disintegrating memory controllers, we reverse a historical trend toward greater integration. We are motivated by evolving bottlenecks that make integration less performance-critical, by memory heterogeneity that makes integration more unwieldy, and by new board/module architectures that make new memory organizations more feasible.

Discrete Controllers (Figure 1a). In the past, controllers were placed in a separate on-board logic chipset, not on the processor die. A front-side bus (FSB) connected the processor and the Northbridge. This organization has several drawbacks as delays are incurred on the FSB and in the Northbridge. These delays increase with contention between processors and contention between memory and I/O requests.

Integrated Controllers (Figure 1b). Integration solves these problems by communicating directly with memory devices, eliminating delays and mitigating contention. Each processor has a local memory controller, which does not compete with I/O requests. However, today's integrated controllers face daunting scalability challenges. These controllers communicate with memory devices over parallel, multi-drop DDR* buses, which do not scale given pin constraints. For example, integrated DDR3 controllers would have difficulty supporting the 512 data pins needed for eight memory channels.

Buffer-on-board (Figure 1c). To address these technology constraints, high-performance processors use a buffer-on-board architecture [19, 23, 14, 45]. Instead of communicating directly to memory devices, integrated controllers encapsulate DDR3 commands, addresses, and data into packets. The controller sends these packets to an off-chip buffer via serialized, point-to-point links that are narrow and fast. The buffer deserializes received packets and relays their contents to memory devices. By replacing wide, parallel buses with narrow, serial links, processors have higher bandwidth to memory with the same pin count.

This organization also avoids most of the latency in discrete controllers. Because each processor has its own buffer dedicated to memory requests, intra-processor and memory-I/O contention are mitigated. However, serial links and buffers introduce a new type of latency that is comparable to the former Northbridge propagation delay. Thus, buffers mitigate pin constraints but negate, in part, the original delay reductions from controller integration.

Disintegrated Controllers (Figure 1d). Our strategy to disintegrate controllers is inspired by buffer-on-board. Since buffers obscure the latency advantages of full controller integration, we propose partial integration and partition controller functionality for modularity. As in a buffer-on-board, we use serial, point-to-point links for higher off-chip bandwidth. But instead of implementing protocol commands, the on-chip controller only packetizes memory requests and pushes the protocol implementation (e.g., DDR*, LPDDR*) off the chip.

By architecting significantly different roles for on- and off-chip controllers, we accommodate heterogeneous memory technologies. We refer to the on-chip controller as the master and the off-chip controller as the slave. The on-chip master provides a homogeneous interface, receiving memory requests and relaying them to slaves according to a certain policy. The off-chip slaves implement heterogeneous protocols, issuing technology-specific commands to its memory devices.

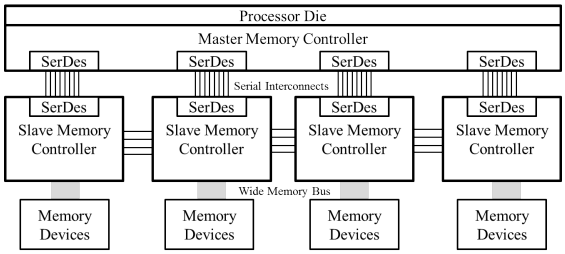

Figure 2: Partitioning controller functionality.

2.2. Partitioning Controller Functionality

Master. As shown in Figure 2, the master controller is integrated with the processor. It receives memory requests from processor caches and forwards them to appropriate slave memory controllers. The relaying policy could be address-based or could account for specific memory technology characteristics. For communication going off-chip, the master serializes memory requests into a packet and sends that packet to the slave via a serial, point-to-point link. For communication coming on-chip, the master receives a packet from the link, deserializes the packet, and relays data to the processor caches.

Our master controller is a simple device that consists only

of an address decoder, serializer/deserializers (SerDes), and queues. By reducing the role of the master and shifting the command protocol to off-chip slaves, this system can extensibly support heterogeneous technologies. With an integrated controller, protocols must be identified and implemented during processor design. In contrast, by separating memory protocol implementation from processor implementation, architects have greater flexibility when deploying technology mixes that best support diverse applications.

Slave. The slave memory controller receives generic memory requests from the master and follows a technology-specific protocol (e.g., DDR*, LPDDR*, LPDDR*-N) when issuing commands to memory devices. Given queued requests, the controller schedules commands to maximize parallelism while enforcing protocol and timing constraints.

Each slave implements a specific protocol and different slaves may implement different protocols. Since slaves control memory devices connected via shared memory buses, the role of the slave is nearly identical to that of a conventional, integrated memory controller. Slaves and conventional controllers differ only in the link interface. Slaves require SerDes circuitry because they receive packetized memory requests from a master over a fast but narrow, serial link. In contrast, conventional memory controllers receive memory requests directly from the processor cache controller.

Slave-to-Slave Communication. In addition to master-to-slave communication, our architecture provides an on-board interconnect that bypasses the processor. This interconnect may be used for page migration between heterogeneous memories. We consider a simple migration protocol.

The master initiates slave-to-slave communication by sending an initiation packet that contains the source address, destination address, and migration granularity. The master is responsible for scheduling the migration packet amongst all other memory requests, including those from the processor.

Upon receiving a migration packet, the slaves perform the required communication according to technology-specific protocols. For example, migrating pages from DRAM to PCM would require one slave to issue reads in the DDR* protocol and another to issue writes in the LPDDR*-N protocol. When migration is complete, the receiving slave sends a completion packet to the master.

2.3. Analyzing Link Design Implications

Disintegrated controllers affect several characteristics of the memory system. Designed for high-performance computing, the architecture dissipates more link power in exchange for greater link bandwidth, higher system capacity, and lower overall system power.

Bandwidth. Fast, serial interfaces between master and slave increase bandwidth for a given number of pins. For example, the IBM Power7 suggests a memory pin-out on a large processor die is 448 [23]. A serial interface uses differential signaling, requiring two pins to manage a single link. If maximum throughput on a serial link is 6.4(Gbps), 448 pins provides a peak theoretical bandwidth of 224×6.4(Gbps)=179.2(GB/s).

In contrast, an integrated memory controller communicating directly with memory devices can use 448 data pins to support up to seven parallel buses. Seven DDR3-1600 channels provide an aggregate theoretical peak of 7×12.8(GB/s)= 89.6(GB/s). Thus, at current technologies, a parallel bus supports only half the bandwidth per pin when compared to a serial link.

Latency. However, these serial links introduce a new latency overhead. Neither the transmitter nor the receiver can process the packet until it receives the whole packet.

$$\text{Delay} = \frac{\text{Packet Size}}{\text{Bandwidth}} = \frac{\text{Packet Size}}{[\#\text{ of Links}]\times[\text{Link Bandwidth}]}$$

For example, consider sending 72B data packets for a read. Given a 36-bit interface with 6.4(Gbps) links, the latency is approximately 2.5(ns). As this delay is added for both transmitter and receiver side, total delay is 5(ns).

However, each slave implements a complete controller, which includes a queue and a scheduler. In steady state, packet serialization and de-serialization latency will be hidden by queuing delay at the slave for most memory accesses. Packet de-serialization for a memory request is not exposed as long as other requests are queued.

Power. Disintegrating the memory controller and linking its master-slave parts with fast, serial interfaces improves bandwidth but incurs a power cost. Although the master's power dissipation decreases due to reduced functionality, the newly introduced slaves dissipate additional power. Master power is now dominated by SerDes circuitry to serialize and deserialize parallel data. Recent SerDes chips consume 9 (pJ/bit) [22]. This cost ranges from 4 to 29 (pJ/bit) depending on the desired data rate and area budget [42]. This cost is incurred twice for SerDes circuitry (i.e., master and slave).

Putting this 2×9(pJ/bit) overhead at the master and slave into perspective, DRAM chips consume more than 200(pJ/bit) [2]. Thus, SerDes may introduce a 10% power overhead. However, this power cost buys a significant increase in off-chip bandwidth. In addition to link interface power, the slave controller dissipates power that is similar to conventional, on-chip memory controllers (e.g., 13.6(W) for a quad-channel controller in a 45nm AMD Opteron [5]).

3. Architecting Rank Parallelism

Although disintegrated controllers enable a heterogeneous memory system, we may encounter challenges when organizing the system to provide the requisite bandwidth or capacity. Emerging resistive memories often incur high write latencies due to expensive programming mechanisms. For example, a recently prototyped x16 PCM sustains only 40(MB/s) of write bandwidth [10]. A high-performance memory architecture must take low-bandwidth parts and construct a high-bandwidth system. We present buffer-based architecture that increases memory parallelism and capacity.

3.1. Inserting Buffers

Given limited device bandwidth and performance, memory rank interleaving is an effective way to enhance channel bandwidth. With a large number of ranks on a memory channel, we can hide one rank's latency with operations to other ranks. However, the number of ranks per channel is limited since each device causes an impedance discontinuity on the shared memory bus and degrades signal integrity. We improve signal integrity by applying recent advances in buffers.

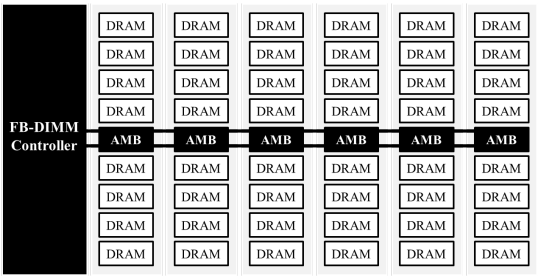

Figure 3: Buffering with Fully-Buffered DIMMs (FB-DIMM)

Fully-buffered DIMM (FB-DIMM). To increase rank count while ensuring signal integrity, a FB-DIMM controller does not communicate directly with memory devices on the parallel bus. Rather, the controller communicates with an advanced memory buffer (AMB), which resides on the memory module (Figure 3). A single FB-DIMM memory channel supports up to 8 AMBs in series.

Although FB-DIMM enables high-bandwidth, high-capacity memory systems, it has two significant disadvantages. First, it incurs large, multi-hop latencies due to serial AMB connections. Second, it incurs large power costs due to heavy-weight AMB functionality, which includes protocol conversion, signal calibration, and packet forwarding.

Load-Reducing Buffer (LR-DIMM). The more recent load-reducing buffer architecture addresses FB-DIMM limitations [38, 18]. Unlike the AMB, LR-DIMM's buffer does not perform protocol conversion. Instead, it simply buffers and re-times signals to enhance signal integrity.

We place an LR-DIMM buffer between the slave and its memory devices. Due to its light-weight functionality, LR-DIMM buffers incur little power overhead relative to registered DIMMs, which are prevalent for server memory [18]. At current technologies, LR-DIMM buffers increase the number of ranks per channel by 4× but adds 5(ns) of delay per hop. We propose new buffer topologies to reduce hop count.

3.2. Buffering Hierarchically

Because FB-DIMMs use point-to-point links to connect buffers, they have little choice in topology; buffers are connected in series. In contrast, we use a parallel bus to connect LR-DIMM buffers, which enables a hierarchical topology (Figure 4). Trees reduce the number of hops to achieve lower end-to-end memory latency.

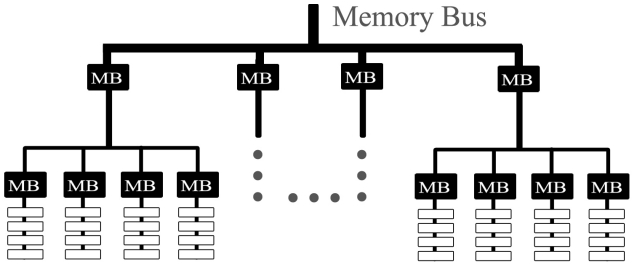

Figure 4: Buffering with hierarchical topology. With 20 buffers (black), 64 memory ranks (white) share a memory bus to the slave.

Such hierarchical memory organizations are more scalable. To attach more than 16 ranks to a channel, we can place buffers in a two-level tree. As shown in Figure 4, the tree has twenty buffers, four in level-1 and sixteen in level-2. These buffers may be placed on either the board or the memory module. For example, level-1 buffers might reside on the board while level-2 buffers might reside on the DIMM.

Multiple Ranks and Heterogeneity. For DRAM, buffers benefit capacity but not bandwidth. At high rank counts, DDR* and high-bandwidth DRAM devices would quickly saturate channel bandwidth. For PCM, however, many devices can share a channel before encountering channel bandwidth constraints. An eight-chip PCM rank provides 8×40(MB/s)=320(MB/s) of write bandwidth and we would need more than 20 PCM ranks to saturate a 6.4(GB/s) LPDDR2-N channel.

Architecting many-rank PCM is complicated by the LPDDR2-N interface, which has no on-die termination (ODT). Without ODT, signal integrity challenges mean un-buffered LPDDR2-N channels can only support the load of four ranks [28]. Each additional buffer allows the channel to support four additional buffers or ranks. For example, a maximally buffered channel supports up to sixteen ranks, using four parallel buffers each driving four ranks. Applied hierarchically, as in Figure 4, a channel can support up to 64 ranks on a single channel.

3.3. Modeling Buffer Power

The buffer has three major functions: improving the signal integrity of clock signals using PLLs, registering input signals, and driving output signals. For each function, we calculate the power dissipation and sum to estimate total buffer power.

We start the calculation for registered memory modules (RDIMMs) and then extend the calculation to our LR-DIMM buffers.

$$\begin{aligned}
\text{Clock PLL} &= I_{dyn-ck} \times f_{ck} \times V_{dd} \times [\#\text{ clock pins}] \\
\text{Registers} &= I_{dyn-reg} \times f_{ck} \times V_{dd} \times [\#\text{ input pins}] \times [\%\text{ activity}] \\
\text{Drivers} &= I_{dyn-drv} \times V_{dd} \times [\#\text{ output pins}] \times (1/2) \times [\%\text{ activity}] \\
\text{Static} &= I_{stat} \times V_{dd}
\end{aligned}$$

From RDIMM datasheets [1], I_{dyn-ck}=68(uA/MHz) for the input clock and $I_{dyn-reg}$=16(uA/MHz) for each data input. Driver current is $I_{dyn-drv}$ = 11(mA) for each data output with

an average 50% duty cycle; drivers do not dissipate power when driving a logical zero. Because RDIMMs buffer clock, command, and address signals, power is dissipated for 2 clock pins and 28 CAS pins. Static current is 20(mA) and supply voltage is 1.5(V).

Consider RDIMM buffers with 25% utilization. Clocks dissipate 163.2(mW), registers dissipate 134.4(mW), drivers dissipate 57.8(mW). Total dynamic power is 355(mW) per buffer. Static power adds another 30(mW).

To calculate power dissipated by our architecture, we extend this analysis for LR-DIMM and LPDDR*-N parameters. To reduce load, LR-DIMMs buffer 108 DQ/DQS signals in addition to the 28 CAS signals; the analysis must account for the higher power of 136 input/output pins. On the other hand, the lower frequencies of LPDDR*-N reduce buffer power. Table 1 shows the net effect.

Table 1: Buffer power for DDR3-1600 and LPDDR2-800, assuming four buffers, each with 25% activity.

	W/buffer	W/channel
DDR3 RDIMM	0.355	1.420
DDR3 LRDIMM	0.741	2.964
LPDDR2 RDIMM	0.207	0.828
LPDDR2 LRDIMM	0.482	1.928

Adding buffers to a tree topology affects power in two ways. If buffers are added in parallel (i.e., widening the tree), the effect on total buffer power is modest. As parallel buffers are added, the number of registers and pins will increase but their utilization will decrease by the same factor. Thus, the net effect on dynamic power is negligible. And dynamic power dominates total power.

However, if we add buffers in series (i.e., deepening the tree), the effect on power is additive. With n-level hierarchical buffering, each signal is buffered n times. And total buffer power dissipation is $n\times$ that of a single buffer. Our architecture uses a quad-tree since each buffer can drive four others without compromising signal integrity in an LPDDR2 interface [28]. And 2-level buffering is sufficient to support up to 64 ranks.

4. Supporting PCM/DRAM Heterogeneity

We apply the proposed architecture to a heterogeneous PCM/DRAM system, which exhibits heterogeneity in several dimensions. First, we encounter performance heterogeneity. PCM latency is approximately $2\times$ that of DRAM and PCM write bandwidth is significantly constrained [26]. However, these PCM disadvantages are offset by extremely low static power dissipation and high cell density. Second, PCM is non-volatile while DRAM is volatile. Finally, the system must accommodate two memory protocols. High-performance DRAM uses DDR3 while PCM uses LPDDR2-N.

The proposed architecture disintegrates memory controllers, adds buffers, and organizes those buffers hierarchically. Disintegrated controllers support DDR3 and LPDDR2-N in an extensible manner; each slave implements the necessary protocol. Buffers enable rank-level parallelism and increase bandwidth for LPDDR2-N PCM. And hierarchical buffer topologies reduce the latency cost of this strategy.

4.1. Navigating Design Objectives

Achieving Target Capacity. Memory system capacity is a function of the memory chips and the channel width:

$$\text{Capacity} = \frac{\text{Capacity}}{\text{Chip}} \times \frac{\text{Bus Width}}{\text{Chip Width}} \times [\text{\# of Ranks}]$$

A memory rank is a set of memory chips that are accessed in parallel to drive a wide data bus. The number of chips per rank is determined by the ratio of bus to chip width. If chips with fewer pins are used, more chips are needed for a given rank width.

For high capacity, a rank should use a larger number of narrower chips. In contrast, for power efficiency, the rank should use fewer, wider chips. Chip count and interface circuitry directly affects DDR3 DRAM background power. At current technologies, both DRAM and PCM are capable of 4-8Gb per chip and 8GB per rank [3, 10]. And to target a particular capacity, we scale the number of ranks. For example, 256GB requires 32 8GB ranks.

In a heterogeneous system, we distribute these ranks among available technologies. For example, a system with 25.6(GB/s) of off-chip bandwidth can support two PCM channels and one DRAM channel, providing 12.8(GB/s) of bandwidth to each technology. We propose an architecture that (i) distributes the requisite PCM ranks across two channels and (ii) adds a few DRAM ranks on a single channel. Because PCM's LPDDR2-N consumes less power than DRAM's DDR3, using PCM to meet the capacity target is efficient. Power-intensive, low-capacity DRAM is used as a cache.

Balancing Bandwidth. By disintegrating the memory controller, we separate the provision of off-chip bandwidth and individual channel bandwidth. The former is determined by master-to-slave serial links and pin count. The latter is determined by slave-to-memory parallel buses and buffer organization. While 32 serial links can support up to 25.6(GB/s) of aggregate master-slave communication, the partitioning of these serial links amongst slaves determines the bandwidth to DRAM versus that to PCM.

Suppose we split 25.6(GB/s) of aggregate off-chip bandwidth equally, providing 12.8(GB/s) to each of the DRAM and PCM slaves. Since DDR3 has a peak bandwidth of 12.8(GB/s), the DRAM slave should support only one such channel. On the other hand, LPDDR2-N has a peak bandwidth of 6.4(GB/s) and the PCM slave should support two such channels.

PCM write bandwidth is of particular interest since it determines the performance of persistent writes, which are imperative for file systems and software checkpointing. PCM chips can sustain 40(MB/s) of write bandwidth [10]. With 16 ranks, each PCM channel supports 5.12(GB/s) of write bandwidth. Note that the peak bandwidth limit for LPDDR2-N is $400\text{MHz} \times 2\text{DDR} \times 8\text{B/bus} = 6.4\text{(GB/s)}$. Thus, we can use

up to 80% of the PCM channel's peak theoretical bandwidth during periods of sustained writes.

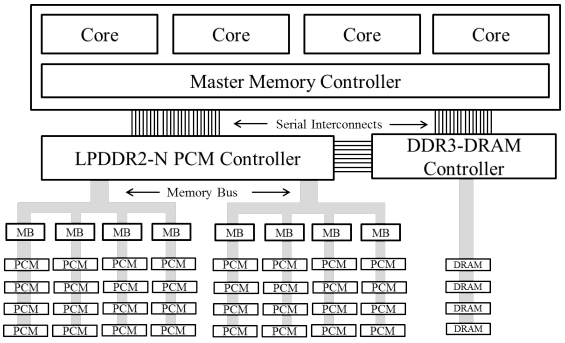

Figure 5: PCM/DRAM Heterogeneity. Memory controllers are disintegrated and slaves implement heterogeneous command protocols. Hierarchical buffer topology enables high-capacity PCM channels. DRAM managed as cache for performance.

Architecting Heterogeneity. Figure 5 presents the heterogeneous PCM/DRAM system. The on-chip master relays memory requests from the core's last-level caches to one of two slaves. One slave controls PCM and supports the LPDDR2-N protocol on two channels with a total of 8 buffers and 32 PCM ranks. The other slave controls DRAM and supports the DDR3 protocol on a single channel with 4 DRAM ranks. Between the slaves, we insert serial links to support 3.2(GB/s) of slave-to-slave bandwidth.

4.2. Managing Data Movement

In an asymmetric architecture, with large PCM capacity and small DRAM capacity, data management is imperative. If the master controller were to relay memory requests based on address space alone, DRAM would be rarely used and its master-slave bandwidth would be under-utilized. Thus, the master must orchestrate page placement and migration.

Baseline Caching and Migration. Management policy for heterogeneous memory have been previously studied. For example, we might organize DRAM and PCM hierarchically, using a small DRAM to cache 4KB PCM pages [34]. All requests query DRAM first. Upon a page miss, the needed page is fetched from PCM. When a page is evicted from the DRAM cache, only its dirty blocks are written back to PCM.

This policy works well if the working set size is comparable to the DRAM cache size. However, it can generate intolerable migration traffic for workloads with larger working sets. Handling a miss requires reading 4KB from PCM and writing it into DRAM. An eviction would further require reading up to 4KB from DRAM and writing it into PCM. Up to 16KB of read and write activity on memory ranks could be generated due to a miss in the last-level cache for 64B of data. Migration traffic could exceed useful memory request traffic.

Alternatively, we might track page popularity in the memory system to migrate hot pages into DRAM and cold pages into PCM [36]. Unlike an inclusive cache policy, migration policies require address re-mapping. Unfortunately, several 4KB pages tend to become hot at the same time, requiring bursts of heavy migration traffic and producing imbalances in PCM and DRAM channel utilization.

Fine-Grained DRAM Caching. These difficulties in migration traffic arise from the large 4KB DRAM cache blocks. To enable fine-grained DRAM caching, we must manage cache tag overheads and cache fill granularity. We begin with prior work for large DRAM caches and extend it with dynamic granularity prefetch and read/write policies.

Consider a DRAM cache with 4KB pages, each comprised of sixty-four 64B blocks, and a 40-bit address space. A 1GB DRAM cache with 4KB blocks require 875KB of tag. The same 1GB cache with 64B blocks requires 68MB of tag, which makes such a naive approach prohibitively expensive. To reduce tag overheads, we rely on MissMaps [27], which hold cache tags at 4KB page granularity but enable 64B block management by associating with each page a vector of block-valid bits (e.g., 64 bits to track 64B blocks in a 4KB page).

We have sized our MissMap to maximize the performance benefits of a 1GB cache [27]. Our MissMap contains 2.875MB of tag and valid bits. These tags are managed by the master controller and occupy the processor die area that is saved by disintegrating the memory controller and shifting protocol functionality off-chip to slaves. A few MB of state is a modest cost for performance in a high-capacity memory system.

Dynamic Granularity DRAM Caching. We extend MissMaps for dynamic granularity prefetch to improve bandwidth utilization. The master dynamically tunes the number of fetched blocks instead of fetching all blocks in the page after a page miss. If a page experiences successive accesses to different blocks, the master increases block prefetch for this page. On the other hand, if a page experiences few block misses, the master decreases prefetch.

To implement dynamic granularity prefetch, we add a three-bit counter to each page's MissMap entry. If the counter has value i for a page, the master fetches 2^i blocks when the master hits this page but misses a block within it. The counter is initialized to $i = 3$ upon a page's first access. The counter increments for every page hit that misses the desired block. Counters decrement at a regular period (e.g., 500(ns)).

Although the DRAM channel enhances performance, we must manage contention and moderate requests to it. The master throttles DRAM use by counting the number of operations to each rank. If more than n operations are sent to a rank within time t, the master controller decrements prefetch counters for all ranks. For example, n is 64 operations and t is 3648ns, which is determined by the latency of those n operations: $t = n \times (tRC + BL/2)$.[1] Moreover, to avoid rapid and de-stabilizing changes in prefetch counters, the controller

[1] tRC is delay between read and subsequent command. $BL/2$ is transfer time determined by burst length divided by 2 in DDR to count cycles.

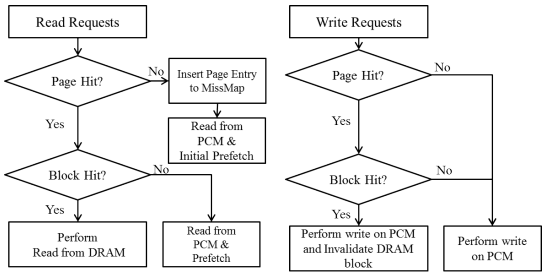

Figure 6: Management for fine-grained DRAM cache. Flow implemented by master controller.

imposes a minimum interval between throttling decrements. Only one such decrement can happen every 250(ns).

Read-only DRAM Caching. Because the last-level processor cache has already exploited much of the available 64B locality, a block evicted from processor caches may have little chance of re-use. This behavior and the desire to reduce migration overheads motivate read-only DRAM caches (Figure 6).

Given a write miss, the master uses a no-write-allocate policy. Given a write hit, the master memory controller directs the block write to PCM and invalidates the DRAM cache block by setting the appropriate MissMap bit. Because DRAM is never written, invalidations are sufficient. Evictions and expensive data migration to PCM are not required.

This architecture may require additional PCM reads for particular read-write-read sequences. An address might be read into DRAM, written into PCM and invalidated in DRAM, and then re-read into DRAM. Had the DRAM cache accommodated writes, the second read into DRAM from PCM would not have been necessary. But this second read is relatively inexpensive; PCM read bandwidth is abundant relative to its write bandwidth.

Finally, read-only DRAM caching supports system applications of PCM non-volatility, like file system consistency [13] and high-performance checkpointing [17]. If the file system writes directly to persistent storage, DRAM-PCM coherence and file system consistency is ensured. And safety improves since file system data becomes persistent more quickly; data is not buffered in volatile DRAM. Software checkpointing no longer performs bulk copies of DRAM pages into PCM. Instead, checkpoints can be distributed in time as individual writes occur during program execution. Combined with mechanisms for PCM fault tolerance [39, 44], these capabilities improve system resilience.

Endurance Implications. With a read-only DRAM cache, all memory writes are directed to PCM, which may seem to contravene conventional wisdom. But cache blocks are less often re-used after last-level cache eviction, which means processor SRAM caches have already exploited opportunities for write coalescing. Writing multiple times to the same block in the DRAM cache, the case in which using read-only DRAM would harm PCM endurance, is not common.

For writes that are required, however, architectural solutions and technology trends mitigate endurance. Even without DRAM, Lee et al. find that associative page buffers in PCM chips can coalesce writes and produce years of lifetime. Such buffers, with differential writes, have recently been implemented in hardware prototypes [11]. Many other mechanisms have been proposed to reduce and level writes [9, 46, 33]. Qureshi et al. provide a good survey [35].

Finally, these architectural mechanisms demonstrate years of lifetime assuming $10^7 - 10^8$ write cycles per cell [26, 33]. More recent prototypes are capable of 10^{11} writes and project future devices capable of 10^{15} writes [24]. With such technology advances, lifetimes extend to tens or hundreds of years.

5. Experimental Methodology

We use Marss86 [31] and modify DRAMSim2 [37] for cycle-accurate processor and memory simulation. Marss86 simulates x86 instructions in full system simulation. DRAMSim2 simulates DDR3-based DRAM and LPDDR2-N-based PCM. We use technology parameters from a recent PCM prototype [10], which achieves a peak write bandwidth of 40(MB/s) per chip. Table 2 summarizes simulation parameters for a 4-core multiprocessor and 25.6(GB/s) of chip-edge bandwidth.

Table 2: Architectural simulation parameters

CPU	Four 2(GHz) 4-way OoO cores 128KB Private L1 Instruction Cache 128KB Private L1 Data Cache	
L2 Cache	Shared 8-way 8MB L2, 64B Cache Line	
Memory Controller	Closed-Page, Queue per Rank, Rank then Bank Round-robin Scheduling	
Technology	PCM	DRAM
Protocol	LPDDR2-N-800, x8	DDR3-1600, x4
Timing	LPDDR2-N[21] t_{RCD}=75(ns)[10]	Micron DDR3[4] t_{RCD} =13.75(ns) t_{CL} =13.75(ns) t_{RP} =13.75(ns)
Cell Read	2.47(pJ/bit) [26]	-
Cell Write	16.82(pJ/bit) [26]	-
IDD Value	LPDDR2-N [21]	Micron DDR3[4]
Rank Write BW	320(MB/s)[10]	12.8(GB/s)

Memory Performance. This experimental methodology differs from trace-driven simulation, which traces last-level cache misses and then feed loads/stores into a memory simulator. Trace-based approaches are fast and easy because it separates the processor and memory. However, without an integrated processor model, trace-driven memory simulation does not fully capture the effects of memory latency increases or of bandwidth contention, which affect processor instruction scheduling.

Another detail we simulate is the constraint on write bandwidth to PCM. Related work limits PCM write bandwidth by simulating a high write latency. However, modern PCMs buffer writes and manage "lazy-writes" within each device [11]. Thus, we limit the number of writes sent to PCM devices, which is more realistic.

Memory Power. Power is determined primarily by the memory protocol and chip interfaces. To achieve the high data rates of DDR3-1600, DRAM interfaces implement expensive link circuitry (delay-locked loops - DLLs, on-die termination -

Table 3: Workload characterization

	Workload	MPKI	Working Set(MB)
WD1	leslie3d leslie3d mcf mcf	31.16	198.9
WD2	lbm leslie3d libquantum mcf	38.38	404.1
WD3	lbm lbm libquantum libquantum	38.25	575.6
WD4	bwaves leslie omentpp sphinx3	11.70	302.6
WD5	GemsFDTD libquantum milc zeusmp	7.24	248.6
WD6	GemsFDTD libquantum milc milc	6.31	181.6
WD7	bzip libquantum milc omnetpp	4.69	129.0
WD8	cactusAMD gcc gobmk zeusmp	2.97	110.7
WD9	astar gobmk hmmer soplex	0.43	23.9

ODT), which incur large static power costs. In contrast, PCM latencies limit data rates, eliminating the need for DLLs and ODT. For this reason, LPDDR2-N dissipates little background power. Moreover, PCM non-volatility obviates refresh. Although expensive PCM programming increases core energy relative to those of DRAM [26], interface and refresh energy dominate in high-capacity systems.

We simulate all power modes. In our experiments, background power increases less than linearly with the number of ranks, indicating that power modes are invoked. But by interleaving data across many ranks for bandwidth, opportunities for rank power-down are constrained in high-performance systems[29]. With extensive operating system or memory controller support, pages may be consolidated into a few active ranks, allowing other ranks to power-down [25].

Workloads. We select memory-intensive workloads from SPEC CPU2006. We simulate 800 million instructions after fast-forwarding 8 billion instructions. The nine multi-programmed sets have different characteristics, as shown in Table 3. WD1-3 have a high number of last-level cache misses per thousand instructions (MPKI) and are memory-bound. WD4-7 have medium MPKI and WD8-9 have small MPKI.

To accurately evaluate many-rank effects on parallel bandwidth and migration traffic, we study four-rank DRAM as a cache for many more ranks of PCM. For experimental purposes, we shrink DRAM capacity to 128MB and stress test our migration policies. This methodology emphasizes bandwidth and neglects potential advantages from fewer page faults.

6. Experimental Evaluation

We first evaluate the scalability of homogeneous many-rank channels. Each rank is comprised of either DDR3 DRAM or LPDDR2-N PCM. With hierarchical buffers, channels support capacity that ranges from 4 ranks to 32 ranks. The PCM data indicates the feasibility of high-capacity, high-bandwidth memory systems that scale with tractable power costs.

6.1. Many-Rank Channels

Performance. As the number of memory ranks increase, greater rank-level parallelism translates into higher sustained bandwidth. Access latency to a rank can be hidden by those to others. Figure 7 quantifies these benefits for PCM and DRAM.

As seen in high-MPKI workloads, rank-level parallelism benefits PCM more since its bandwidth starts from a lower base due to higher device latencies. For example, in WD3, sustained PCM and DRAM bandwidth increase by 65% and 16%, respectively, as rank count increases from 4 to 32.

But of course, bandwidth increases only if memory requests have been queued and access different ranks. When PCM bandwidth is limited by device delay rather than queuing delay, as in WD6-9, additional ranks simply increase capacity without benefiting data transfer rates.

Power. Many-rank channels incur a power cost for their capacity and performance. And these costs increase rapidly for a homogeneous DRAM architecture. In contrast, PCM power costs are far more attractive as capacity increases. PCM's advantage over DRAM arises from its low static power dissipation, which is an increasingly large fraction of the total in high-capacity memory systems.

Consider a high-MPKI workload like WD3. In Figure 8, DRAM background and refresh power comprise 50% of the total in a 4-rank system. But this cost increases to more than 75% in a 32-rank system. At high capacities, these DRAM power costs are an efficiency bottleneck.

In contrast, PCM power scales far more slowly with capacity. This efficiency arises from its device interface and its non-volatility. As a technology with higher latencies, PCM operates its channels at lower frequencies. At these lower data rates, PCM devices do not require power-intensive DLLs and ODT circuitry, which accounts for much of the background power in high-performance DRAM. And as a resistive memory, PCM does not require refresh. These effects keep static power tractable as PCM capacity increases to 32 ranks.

Energy. Figure 9 presents the energy per bit transferred. As capacity increases, PCM energy is flat. In contrast, DRAM energy increases rapidly as its increasing background and refresh power is amortized over the same number of data transfers. These effects are particularly problematic for low-

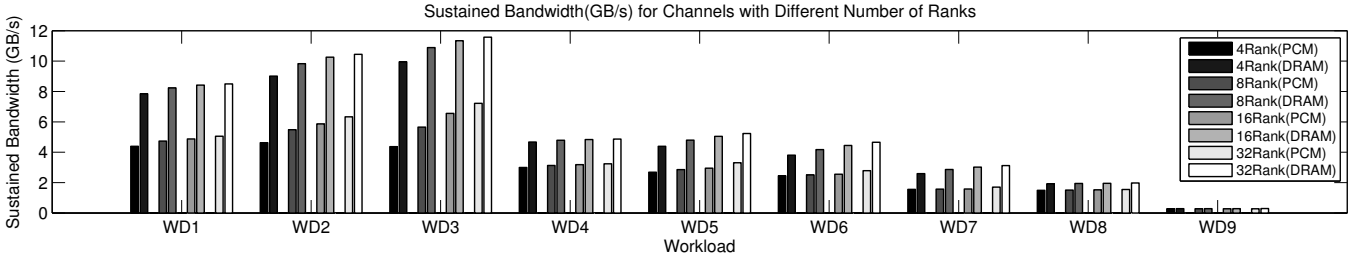

Figure 7: Sustained memory bandwidth as rank count increases, which illustrates rank-level parallelism.

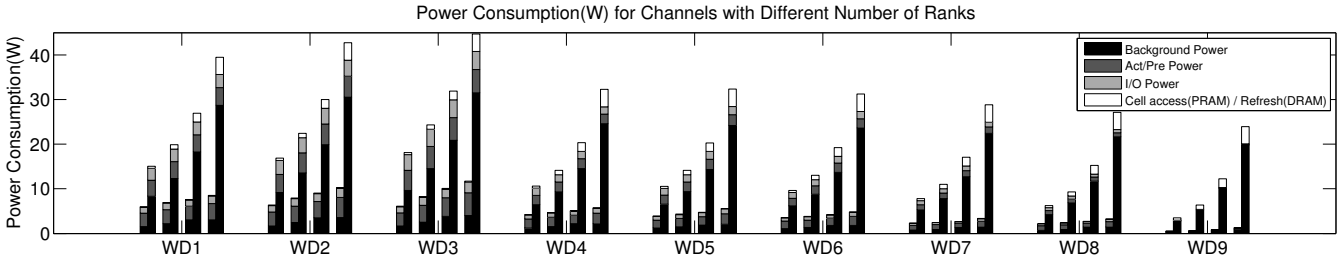

Figure 8: Power as rank count increases. As in Figure 7, for each workload, bars show nRank(PCM) followed by nRank(DRAM), $n \in [4, 8, 16, 32]$.

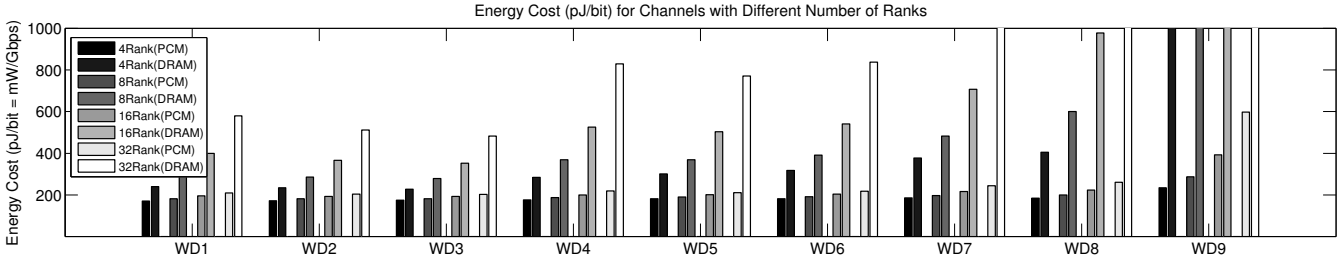

Figure 9: Energy as rank count increases.

MPKI workloads, which perform few transfers.

Alternatively, consider high-MPKI workloads WD1-3. To fetch a 32-bit word, high-capacity DRAM consumes up to 600(pJ/bit) or 19.2(nJ/word). To put these numbers into perspective, a high-performance processor consumes 20(nJ/inst) [15]. And an instruction may require several words of data. As capacity increases, the costs of DRAM data movement would dwarf those of executing an instruction.

In contrast, PCM consumes 200(pJ/bit) or 6.4(nJ/word). And these costs remain relatively flat as capacity increases and as memory activity varies. Unlike DRAM, PCM consumes energy in proportion to activity regardless of capacity. Thus, PCM is attractive for high-capacity memory systems.

For perspective, note that energy per bit is calculated by dividing power dissipated by utilized bandwidth (mW/Gbps). DRAM energy per bit tends to be large when bandwidth utilization is low; static power is a large fraction of the total and static power is amortized over few bits transferred.

This effect is most pronounced in high-capacity, many-rank memory systems since static power increases linearly with the number of ranks while dynamic power is a function of workload (and thus unaffected). As a result, energy per bit in our many-rank setting is larger than that in settings with fewer ranks (e.g., 200 versus 60(pJ/bit)). These numbers can be verified by invoking Micron's power calculator with low channel utilization (e.g., 15%) [2].

6.2. Heterogeneous Memory Architecture

While PCM efficiency is attractive, DRAM may be required to supplement performance. We evaluate our management policy on a PCM/DRAM system (§ 4), comparing it against homogeneous baselines and alternative policies.

- **DRAM (2C16R)** – Homogeneous DRAM with two channels and sixteen ranks each.
- **PCM (2C16R)** – Homogeneous PCM with two channels and sixteen ranks each. Provides an equal channel comparison for DRAM (2C16R) but halves bandwidth.
- **PCM (4C8R)** – Homogeneous PCM with four channels and eight ranks each. Provides an equal bandwidth comparison for DRAM (2C16R) but doubles channel count.
- **Hetero (Base Cache)** – Heterogeneous PCM/DRAM with 32 PCM and 4 DRAM ranks. Manages DRAM as simple cache (§4.2, [34]).
- **Hetero (Base Migrate)** – Heterogeneous PCM/DRAM with 32 PCM and 4 DRAM ranks. Manages DRAM with hot/cold page migration (§4.2, [36]).
- **Hetero (New Cache)** – Heterogeneous PCM/DRAM with 32 PCM and 4 DRAM ranks. Manages DRAM with fine-grained caching, dynamic granularity cache fill, and read-only policies (§4.2).

These systems have the same theoretical peak bandwidth of 25.6(GB/s) to the processor. These systems provide 32 ranks of capacity. In the heterogeneous systems, PCM is supported by 4 ranks of DRAM cache.

Performance. Replacing homogeneous DRAM with homogeneous PCM increases run-time, which we measure as the average delay (cycles) per instruction in Figure 10. Memory-intensive workloads incur 1.6-1.8× delay as LPDDR2-N halves channel bandwidth relative to DDR3 and as PCM increases latencies relative to DRAM.

In cases with sufficient memory-level parallelism, such as WD2-3, distributing PCM ranks across more channels improves performance (2C16R versus 4C8R). More generally, however, a small DRAM cache reduces the PCM penalty most effectively. But we must determine how to manage this

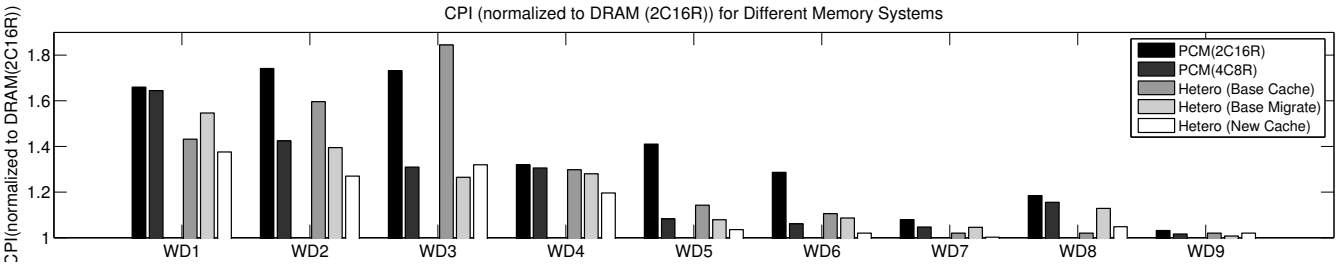

Figure 10: Application delay (cycles per instruction) for alternative memory architectures. Bars show instructions per cycle normalized to homogeneous DRAM(2C16R).

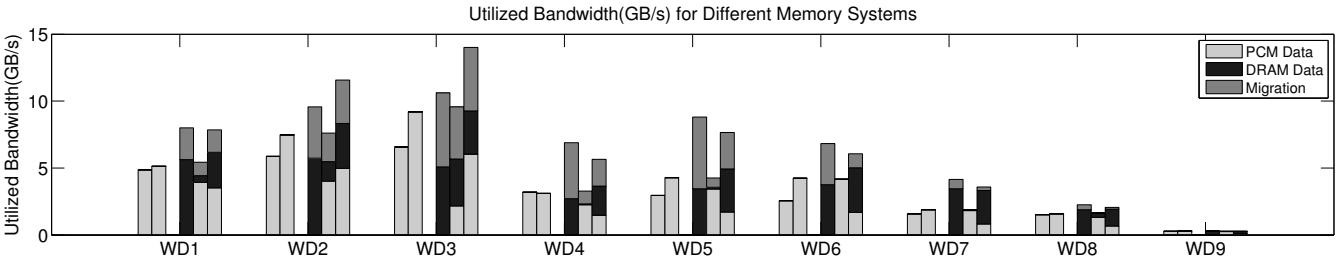

Figure 11: Sustained bandwidth for alternative memory architectures. Bars show PCM (2C16R), PCM (4C8R), Hetero (Base Cache), Hetero (Base Migrate), Hetero (New Cache).

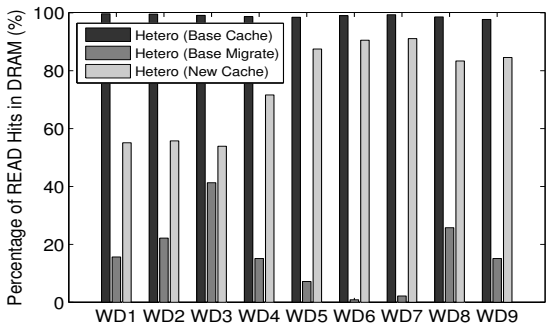

Figure 12: Percentage of reads satisfied by DRAM.

DRAM. With poor management, migration traffic could lead heterogeneous PCM/DRAM to perform worse than homogeneous PCM (e.g., WD3 and base cache).

Base cache/migrate policies have highly variable performance. The base cache policy manages DRAM as an inclusive cache, which performs well for 3 of 9 workloads. The base migrate policy manages DRAM by migrating hot pages into it, which performs well for the other 6 workloads. In contrast, our fine-grained cache policies outperform either or both base policies across the spectrum of workload behaviors. On average, we reduce homogeneous PCM (4C8R)'s performance penalty by 39% when using our new cache policies for heterogeneous PCM/DRAM.

Balancing Bandwidth Utilization. Differences in performance arise from differences in bandwidth utilization. In a heterogeneous system, memory-to-processor data flows between two serial interfaces: PCM master-slave and DRAM master-slave. For performance, we must load-balance PCM and DRAM transfers. Too much traffic on one technology's channel causes contention and under-utilizes the other.

Under an effective management policy, DRAM master-slave links should be well utilized. DRAM has half the total bandwidth to the master (12.8 of 25.6(GB/s) but contributes less than 0.4% of the total capacity (4-rank DRAM versus 32-rank PCM), making locality management imperative yet difficult. Ideally, the relatively small amount of DRAM data should account for half the transfers to/from the processor.

Figure 11 shows master-slave bandwidth for DRAM and PCM ranks. The base cache and migrate policies do not balance load. In the base cache policy, all processor transfers access DRAM. And because it aggressively prefetches 4KB pages from PCM, Figure 12 indicates that 99% of reads hit in DRAM. PCM bandwidth is under-utilized.

Illustrating another extreme, the base hot/cold migration policy does not use PCM much. DRAM serves fewer than 25% of reads. Hot/cold migrates pages carefully but this leads to low DRAM use. In Figure 11, only WD2-3 use DRAM bandwidth in any significant way. Most processor transfers access PCM; DRAM bandwidth is under-utilized.

Our new cache policies are more effective, dividing transfers between PCM and DRAM to utilize both channels. For high-MPKI workloads (WD1-3), our policies throttle DRAM use due to contention. As shown in Figure 12, DRAM transfers account for 55% of the total. The other 45% transfers access PCM. For other workloads, DRAM contention is modest and up to 80% of transfers involve DRAM. Thus, by dynamically adjusting cache fill granularity according to DRAM contention, our policy manages data placement and balances bandwidth utilization across DRAM and PCM interfaces.

Load-balancing memory interfaces requires migrating data between technologies. Figure 11 puts this overhead into perspective, showing PCM/DRAM slave-slave migration band-

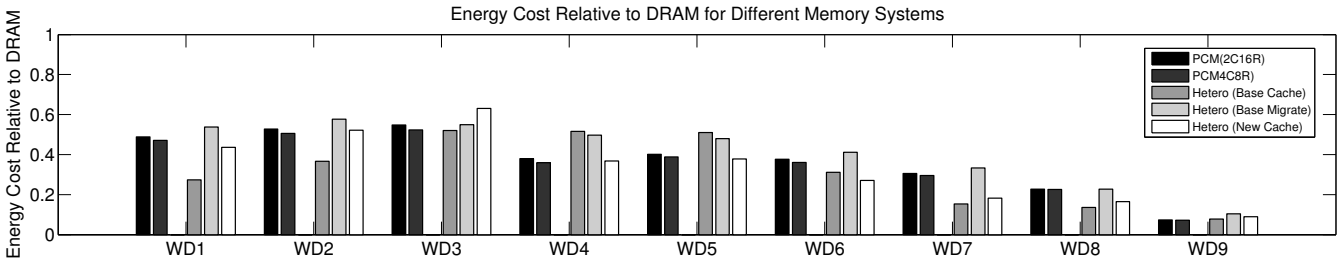

Figure 13: Energy for alternative memory architectures.

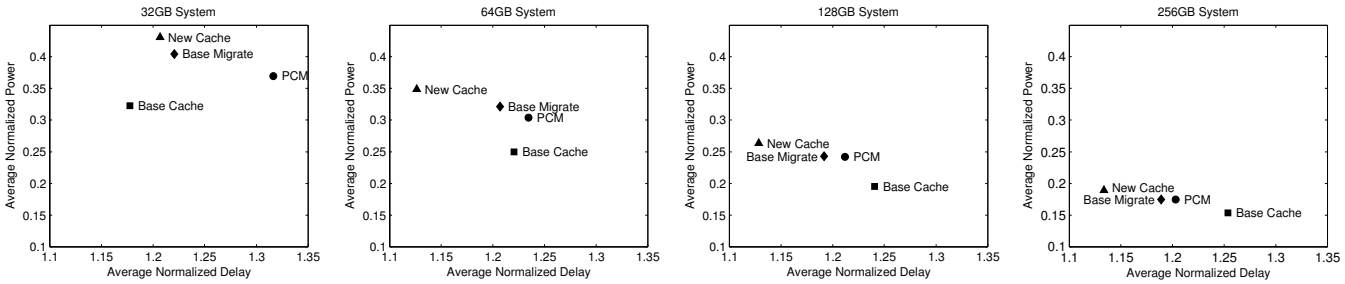

Figure 14: Power-delay trade-offs for alternative memory architectures. Power and delay are normalized to homogeneous DRAM and averaged (geometric mean) across workloads.

width. The base cache policy requires frequent 4KB migrations, producing bursty traffic. Such migrations harm performance (e.g., WD3) as the processor must contend with slave-slave transfers at the DRAM ranks. And in several cases, migration transfers outnumber useful ones (e.g., WD3-5). Compared to the baselines, our new cache require fewer migration transfers measured as a fraction of the total.

Power and Energy. Figure 13 shows energy per bit transferred, dividing total memory power by the number of bits transferred to the processor. Note that migrated bits are not counted as useful work. PCM with a DRAM cache consumes, on average, 0.35× (or less) the energy of a homogeneous DRAM system, which dissipates high background power. Amongst heterogeneous alternatives, the base migrate policy is less efficient.

6.3. Heterogeneous System Design Space

For various memory capacities, Figure 14 plots the power and delay for architectural alternatives. Data is normalized to a homogeneous DRAM system and averaged across workloads. At the smallest 4 rank capacity, any system with PCM reduces power (0.35×) but incurs large performance penalties (> 1.3×). Power reductions arise from the low static power of PCM. Performance penalties are due to higher latency and lower bandwidth at the PCM device.

Increasing PCM capacity by increasing rank count reduces delay for most design points. The exception is the base cache policy, which serves 99% of memory accesses from DRAM and does not benefit from PCM rank-level parallelism. At large capacities, our new DRAM policies perform better than prior policies. Performance incurs less than a 1.15× penalty as we access both PCM and DRAM in parallel and manage DRAM more effectively as a fine-grained, read-only cache. Power is only 0.2x that of a homogeneous DRAM system. Static and background power are the primary determinants of efficiency. These costs increase with DRAM capacity, causing us to favor high-capacity memory systems that use PCM.

7. Related Work

Managing Heterogeneous Memory. Several heterogeneous memory systems use a small DRAM as a cache for a larger PCM-based main memory [7, 16, 30, 34]. Ramos et al. suggested a management policy that migrates hot pages to DRAM while keeping cold pages in PCM[36]. Phadke et al. profile application memory accesses for applications and place working sets into particular heterogeneous DRAM modules, matching application-specific demands to a module's power efficiency, latency, or bandwidth [32]. It employs off-line profiling while we implement dynamic mechanisms.

Our fine-grained read-only DRAM caching policy outperforms previous management policies by reducing migration traffic while maintaining an appropriate DRAM utilization level. Migration-induced contention could be mitigated at the device-level with staged reads, which add registers to DRAM chips to exploit bank parallelism [8]. In contrast, we mitigate contention at the system-level, balancing bandwidth utilization across channels and technologies.

Using Heterogeneous Memory. Condit et al. present a new file system that improves data safety and consistency with phase change memory [13]. Dong et al. propose stacking PCM on DRAM for high-bandwidth HPC checkpointing [17]. These systems could be implemented in our architecture.

Coburn et al. and Volos et al. separately exploit heterogeneous non-volatile memory for persistent data structures by

providing programmer-exposed primitives and abstractions [12, 41]. Such user-level libraries for distinguishing allocations to DRAM and PCM are not required in our architecture since all allocations are made to PCM and cached in DRAM.

Hybrid Memory Cube. Recent studies in 3D integration [20] contrast withour approach to disintegrated controllers. HMC is made for 3D-stacked DRAM connected to the processor die via TSV interconnects. Packaging all of these dies together, HCM cannot disintegrate memory controllers from the processor and memory technology. Indeed, HCM promotes greater integration and less flexibility, while we promote less integration and more flexibility.

Emerging Technologies. While we double per pin bandwidth by using serial links, photonics might further increase bandwidth using dense wavelength-division multiplexing [6]. Udipi et al. shift protocol implementation to a control die in 3D-stacked memory. Because stacks' controllers share a bus, a reservation protocol is required [40]. In contrast, we propose disintegrated slave controllers with point-to-point links to the processor, obviating the need for reservations.

8. Conclusion

We present a scalable heterogeneous memory architecture that utilizes both DDR3 DRAM and LPDDR2-N PCM technologies. We implement this architecture with disintegrated memory controllers, with buffers for signal integrity, and with hierarchical rank organization for capacity. Data movement is expensive and we present DRAM cache management policies that provide performance and energy efficiency.

Acknowledgements

This work is supported, in part, by NSF grant CCF-1149252 (CAREER) and a Google Faculty Research Award. This work is also supported by STARnet, a Semiconductor Research Corporation program, sponsored by MARCO and DARPA. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of these sponsors.

References

[1] "1.3V/1.5V registering clock driver with parity test and quad chip select," IDT, Tech. Rep. SSTE32882HLB.
[2] "Calculating memory system power for DDR3," Micron, Technical Note TN-41-01, 2007.
[3] "DDP 4gb B-die DDR3 SDRAM specification," Samsung, Tech. Rep. K4B4G0446B, 2009.
[4] "DDR3 SDRAM Datasheet," Micron, Tech. Rep. MT41J256M4, 2009.
[5] AMD, "ACP-The truth about power consumption starts here," Tech. Rep. AMD-43761C, 2009.
[6] S. Beamer *et al.*, "Re-architecting DRAM memory systems with monolithically integrated silicon photonics," in *ISCA*, 2010.
[7] A. Bivens *et al.*, "Architectural design for next generation heterogeneous memory systems," in *International Memory Workshop (IMW)*, 2010.
[8] N. Chatterjee *et al.*, "Staged reads: Mitigating the impact of dram writes on dram reads," in *HPCA*, 2012.
[9] S. Cho and H. Lee, "Flip-n-write: a simple deterministic technique to improve pram write performance, energy and endurance," in *MICRO*, 2009.
[10] Y. Choi *et al.*, "A 20nm 1.8V 8Gb PRAM with 40MB/s program bandwidth," in *ISSCC*, 2012.
[11] H. Chung *et al.*, "A 58nm 1.8V 1Gb PRAM with 6.4MB/s program BW," in *ISSCC*, 2011.
[12] J. Coburn *et al.*, "NV-Heaps: Making persistent objects fast and safe with next-generation, non-volatile memories," in *ASPLOS*, 2011.
[13] J. Condit *et al.*, "Better I/O through byte-addressable, persistent memory," in *SOSP*, 2009.
[14] E. Cooper-Balis *et al.*, "Buffer-on-board memory system," in *ISCA*, 2012.
[15] W. Dally *et al.*, "Efficient embedded computing," *IEEE Computer*, 2008.
[16] G. Dhiman *et al.*, "PDRAM: A hybrid PRAM and DRAM main memory system," in *DAC*, 2009.
[17] X. Dong *et al.*, "Leveraging 3D PCRAM technologies to reduce checkpoint overhead for future exascale systems," in *SC*, 2009.
[18] Inphi, "Basics of LRDIMM," http://www.edn.com/article/519386-Basics_of_LRDIMM.php, 2011.
[19] Intel, "Intel 7500 scalable memory buffer datasheet," 2011.
[20] J. Jeddeloh and B. Keeth, "Hybrid memory cube new dram architecture increases density and performance," in *VLSI Technology (VLSIT), 2012 Symposium on*, june 2012, pp. 87 –88.
[21] JEDEC, "Low power double data rate 2 (LPDDR2)," 2011.
[22] A. Joy *et al.*, "Analog-DFE-based 16Gb/s SerDes in 40nm CMOS that operates across 34dB loss channels at Nyquist with a baud rate CDR and 1.2Vpp voltage-mode driver," in *ISSCC*, 2011.
[23] R. Kalla, B. Sinharoy, W. Starke, and M. Floyd, "Power7: IBM's next-generation server processor," *IEEE Micro*, vol. 30, no. 2, 2010.
[24] I. Kim *et al.*, "High performance PRAM cell scalable to sub-20nm technology with below 4F2 cell size, extendable to DRAM applications," in *VLSI*, 2010.
[25] A. Lebeck *et al.*, "Power aware page allocation," in *ASPLOS*, 2000.
[26] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting phase change memory as a scalable dram alternative," in *ISCA*, 2009.
[27] G. Loh and M. Hill, "Efficiently enabling conventional block sizes for very large die-stacked DRAM caches," in *MICRO*, 2011.
[28] K. Malladi *et al.*, "Towards energy proportional datacenter memory with mobile DRAMs," in *ISCA*, 2012.
[29] D. Meisner *et al.*, "Power management of online data-intensive services," in *ISCA*, 2011.
[30] H. Park *et al.*, "Power management of hybrid DRAM/PRAM-based main memory," in *DAC*, 2011.
[31] A. Patel *et al.*, "MARSSx86: A Full System Simulator for x86 CPUs," in *DAC*, 2011.
[32] S. Phadke and S. Narayanasamy, "MLP aware heterogeneous memory system," in *DATE*, 2011.
[33] M. Qureshi *et al.*, "Enhancing lifetime and security of PCM-based main memory with start-gap wear leveling," in *MICRO*, 2009.
[34] ——, "Scalable high performance main memory system using phase-change memory technology," in *ISCA*, 2009.
[35] ——, *Phase change memory: From devices to systems*. Morgan Claypool, 2011.
[36] L. Ramos *et al.*, "Page placement in hybrid memory systems," in *ICS*, 2011.
[37] P. Rosenfeld *et al.*, "Dramsim2: A cycle accurate memory system simulator," *IEEE Computer Architecture Letters*, pp. 16–19, 2011.
[38] Samsung, "Samsung DDR3 LRDIMM," http://www.samsung.com/global/business/semiconductor/support/brochures/downloads/memory/samsung_LRDIMM.pdf, 2010.
[39] S. Schechter *et al.*, "Use ECP, not ECC, for hard failures in resistive memories," in *ISCA*, 2010.
[40] A. Udipi *et al.*, "Combining memory and a controller with photonics through 3d-stacking to enable scalable and energy-efficient systems," in *ISCA*, 2011.
[41] H. Volos *et al.*, "Mnemosyne: Lightweight persistent memory," in *ASPLOS*, 2011.
[42] R. Williams *et al.*, "Server memory road map," Presentation: Server Memory Forum, 2011.
[43] W. A. Wulf and S. A. McKee, "Hitting the memory wall: implications of the obvious," *SIGARCH Comput. Archit. News*, vol. 23, no. 1, pp. 20–24, Mar. 1995.
[44] D. Yoon *et al.*, "FREE-p: Protecting non-volatile memory against both hard and soft errors," in *HPCA*, 2011.
[45] H. Zheng *et al.*, "Decoupled DIMM: Building high-bandwidth memory system using low-speed DRAM devices," in *ISCA*, 2009.
[46] Zhou *et al.*, "A durable and energy efficient main memory using phase change memory technology," in *ISCA*, 2009.