# Designing Scalable Heterogeneous Memory for High-Performance Computing

Tae Jun Ham

Advisor : Benjamin C. Lee

December 1st, 2011

**Abstract**

As different DRAM memory protocols and emerging memory technologies present themselves as competitive alternatives to current DDR3-based HPC memory systems, the need for a memory system that can handle heterogeneity increases. To satisfy this need, this paper proposes an architecture for a scalable, heterogeneous memory system based on hierarchical memory controllers. In addition, use of multi-level memory buffer is also introduced to provide better memory system bandwidth and capacity. In the end, we demonstrate the effectiveness of this system with a case study: DRAM- and PRAM-based heterogeneous memory for HPC checkpointing.

## 1 Introduction

Recent advances in emerging memory technologies and new DRAM memory protocols are providing attractive alternatives to current DDR3-based memory systems in high-performance computing (HPC). Phase change memory may be viable as a DRAM-alternative, providing new safety and consistency properties in high-performance systems [7, 20, 26, 32]. In addition, Kozyrakis suggests the use of LPDDR2 DRAM memory in servers for its energy proportionality[19]. Each of these technologies provide some advantage over DDR3.

Since no single memory technology or protocol dominates all others, heterogeneous memory systems are required to fully exploit these diverse memory technologies. While heterogeneous memory systems have been studied in the past (e.g., [24, 6, 11]), especially in the context of 3D stacking and through-silicon-vias, we propose a new architecture that organizes memory chips using currently available technologies.

First, we propose a heterogeneous memory system that disintegrates memory controllers from the processor, reversing a recent trend toward on-chip controllers. Discrete memory controllers provide extensibility and flexibility, separating processor design choices from those in memory. A disintegrated memory controller has a master and slaves.

The on-chip master, which is integrated with the processor, does not issue commands. Instead, it simply forwards memory requests from the last-level processor cache to protocol-specific, off-chip slaves via narrow, serialized point-to-point links. Each slave memory controller communicates with memory devices using the appropriate protocol, which is the traditional role of on-chip integrated memory controllers. This architecture is scalable and extensible; we can architect several different slave controllers without being limited by processor area or pin count.

Second, we propose hierarchical memory buffers to cope with low bandwidth in some heterogeneous memory technologies and to increase capacity by adding ranks to a single channel. Furthermore, to demonstrate the effectiveness of the architecture, we apply a heterogeneous DRAM/PRAM architecture to low-overhead, high-performance checkpointing. Our contributions include:

- Using disintegrated memory controllers, we present an extensible heterogeneous memory architecture (§2)
- Using multi-level memory buffers, we present a high-bandwidth, high-capacity memory architecture (§3)

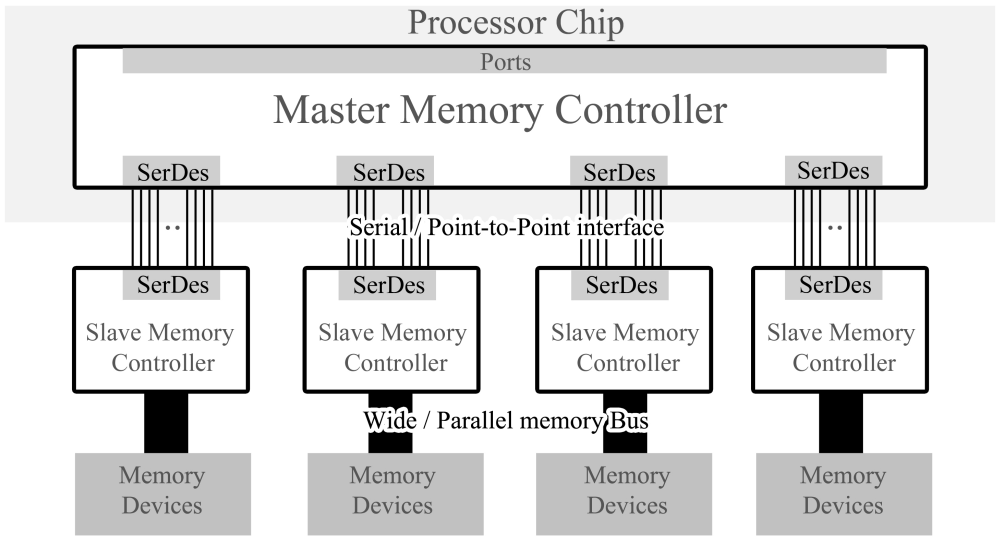

Figure 1: Heterogeneous Memory Architecture : On-chip master receives memory requests, relays them to off-chip slaves via point-to-point links. Off-chip slaves are heterogeneous and receive requests, apply technology-specific protocols, and communicate to memory devices.

- Applying a DRAM/PRAM-based heterogeneous system, we demonstrate low checkpointing overheads (§4)

## 2 Heterogeneous Controllers

Integrated, on-chip memory controllers are favored in high-performance processors for their low latency. However, integrated controllers that directly communicate with memory devices over wide, parallel, multi-drop DDR* buses cannot scale given constraints in area, power, and pin-out. For example, integrated DDR3 controllers have difficulty supporting eight channels, which would require more than 512 pins.

### 2.1 Disintegrated Memory Controllers

To address technology constraints, high-performance processors such as the Intel Xeon 7500 [15] and the IBM Power7 [18]) utilize a buffer-on-board architecture. Integrated memory controllers do not directly communicate with memory devices. Instead, on-chip controllers encapsulate DDR3 commands, addresses, and data into packets and send these packets to an off-chip buffer-on-board via serialized, point-to-point links that are narrow and fast. The off-chip buffer-on-board then de-serializes received packets and relays their contents to memory devices. Thus, processors can use fast, serialized links to maintain bandwidth but use fewer pins.

Our strategy to disintegrate controllers is inspired by buffer-on-board architectures. We use serial, point-to-point interfaces to achieve higher bandwidth. But we architect significantly different roles for on- and off-chip controllers. 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. The off-chip slaves implement heterogeneous protocols, issuing specific protocol commands to its memory devices.

### 2.2 Hierarchical Memory Controllers

The hierarchical memory architecture is shown in Figure 1. As shown, the system is composed hierarchically with a master and heterogeneous slaves.

**Master Memory Controller.** The master controller is integrated with the processor. It receives memory requests from processor caches. The master forwards requests to appropriate slave memory controllers based on the mapped address; it maps specific technologies to specific address spaces. 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.

Our architecture significantly reduces the role of on-chip memory controllers. Conventional, integrated memory controllers are responsible for scheduling and issuing protocol-specific DRAM commands. In contrast, our master controller is a simple device that consists only of an address decoder, serializer/deserializers (SerDes), and queues or buffers at the interfaces to the cache and slave controllers.

By reducing the role of the master and shifting the responsibility of protocol commands to off-chip slaves, this system can extensibly support heterogeneous technologies. With a conventional, integrated controller, protocols must be identified and implemented during processor design. In contrast, by separating memory protocol implementation from processor implementation, system architects have greater flexibility when deploying the heterogeneous mix of technologies that best supports application needs.

**Slave Memory Controller.** The slave memory controller receives generic memory requests from the master and follows a technology-specific memory protocol (e.g., DDR*, LPDDR*, LPDDR*-N) when issuing commands to memory devices. In particular, the slave must deserialize packets received from the master over the point-to-point link. Given queued requests, the controller schedules protocol-specific commands to maximize parallelism while enforcing protocol and timing constraints. Read data destined for the processor is serialized and sent to the master.

The slaves communicate directly with memory devices. Each slave implements a specific protocol and different slaves implement different protocols. Since slaves control memory devices connected via multi-drop 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.

## 2.3 Design Analysis

Hierarchical memory controllers affect several characteristics of the memory system. Designed for high-performance computing, the architecture trades power for bandwidth.

**Bandwidth and Latency.** Fast serial interfaces between the master and slave increase bandwidth for a given number of pins. For example, consider the IBM Power7[18], which suggests a memory pin-out on a large processor die is 224. If maximum throughput on a serial link is 6.4(Gbps), bandwidth of $224 \times 6.4$(Gbps)=179.2(GB/s) can be achieved. In contrast, an on-chip memory controller directly communicateing with memoy devices supports a maximum of three data channels in a 224-pin budget. Three DDR3-1600 channels provide an aggregate theoretical peak of $3 \times 12.8$(GB/s)=38.4(GB/s).

However, narrow and serial links introduce a new latency overhead. The receiver cannot process the packet until it receives the whole packet.

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

**Power and Energy.** Disintegrating the memory controller and linking its 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 [17] dissipate approximately 9(pJ/bit = mW/Gbps). This cost can range from 4.3 to 29.2 (pJ/bit) depending on the desired data rate and area budget [31]. This cost is incurred twice for SerDes circuitry (i.e., master and slave).

Putting this number into perspective, DDR3 x4 chips may consume more than 200(pJ/bit) depending on channel utilization [21]. Thus, SerDes may introduce a 10% power overhead. Combined with the bandwidth analysis, we find this power cost buys a significant bandwidth increase.

In addition to link interface power, the slave controller dissipates power that is similar to conventional on-chip memory controllers. Recent quad-channel, integrated memory controllers dissipate 13.6(W) [1].

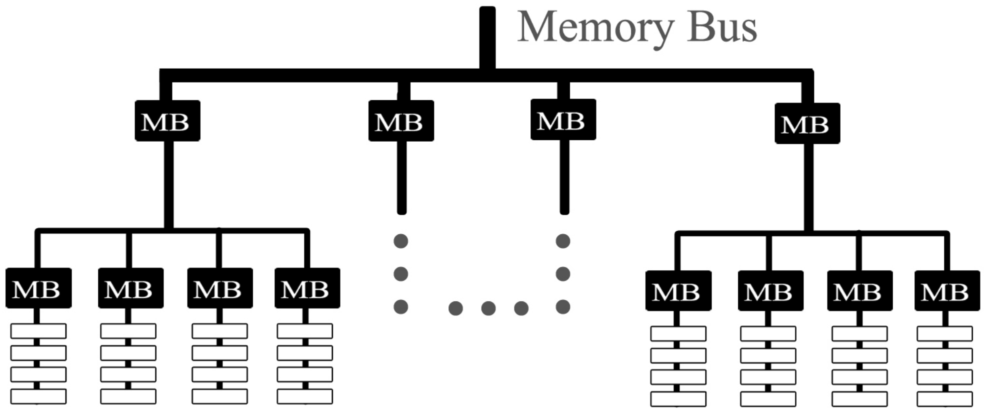

Figure 2: Two-level memory buffering (black) to architect 64 PRAM ranks (white) on a channel.

## 3 Multi-level Buffers

Although discrete 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 PRAM sustains only 6.4(MB/s) write bandwidth[5]. A high-performance memory architecture must take low-bandwidth parts and construct a high-bandwidth system. We present a multi-level buffer architecture that increases memory parallelism and bandwidth.

### 3.1 Load-Reducing Buffers

With limited device bandwidth and performance, an effective way to enhance the total bandwidth is memory rank interleaving. By having a large number of ranks on a memory channel, we can increase channel utilization even with low-bandwidth memory devices. However, the number or ranks per channel is limited since each device causes an impedance discontinuity on memory buses and degrades signal integrity. To cope with this signal integrity challenge as the number of ranks increases, we insert memory buffers.

A memory buffer is a simple device located between the slave memory controller and the memory devices. It buffers all signals destined for the memory channel to enhance signal integrity. Recent DRAM technologies, such as LR-DIMM, places this buffer on the memory module to allow more ranks per DIMM and increase system capacity [29, 14]. Buffering for DRAMs quickly encounters diminishing marginal returns. LR-DIMMs quickly saturate a DDR* memory channel given the high bandwidth of DRAM devices and ranks; channel bandwidth is a bottleneck for many-rank DRAM systems.

When extending the concept of buffers to low-bandwidth PRAM devices, however, we require many PRAM ranks to saturate the same channel bandwidth. With current technology, memory buffers increase by 4× the number of ranks that can be attached on a channel at a cost of 5(ns) delay [29]. And the architecture incurs almost no power overhead relative to registered modules, which are the prevalent module architecture for server memories [14].

In a PRAM buffered architecture, we assume four PRAM ranks share a channel without degrading signal integrity. Although DDR3 protocols allow eight logical ranks per channel [15], PRAM uses an LPDDR2-N protocol that supports fewer ranks per channel. Unlike DDR3, LPDDR2-N does not provide on-die-termination (ODT) to improve signal integrity. Without ODT, un-buffered channels can only support four ranks. With buffering, signal integrity is enhanced. For example, four buffers allow a channel to support sixteen ranks per channel.

Even with four buffers, PRAM write bandwidth underutilizes a channel that provides DDR3-like bandwidth. Thus, for low-bandwidth technologies, we consider the memory-level parallelism of multi-level buffers. For example, Figure 2 illustrates a two-level buffer organization that allows 64 ranks of PRAM per channel. The two-level organization illustrates a total of 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, place level-1 buffers on the board and level-2 buffers on the DIMM.

Table 1: Architectural Simulation Parameters

| | | |
|---|---|---|
| CPU | 2(GHz) 8-way Superscalar<br>32KB L1 Instruction Cache<br>64KB L1 Data Cache | |
| L2 Cache | 8-way 2MB L2, 64B Cache Line | |
| Memory Controller | Closed-Page, Queue per Rank,<br>Rank then Bank Round-robin Scheduling | |
| Technology | PRAM | DRAM |
| Protocol | LPDDR2-N-800, x16 | DDR3-1333, x4 |
| Timing | LPDDR2-N[16],<br>$t_{RCD} = 25(cycle)$[5] | Micron DDR3[22],<br>$t_{RCD}$ =10(cycle), $t_{CL}$ =10(cycle),<br>$t_{RP}$ =10(cycle) |
| Cell Read | 2.47(pJ/bit) [20] | - |
| Cell Write | 16.82(pJ/bit) [20] | - |
| IDD Value | LPDDR2-N [16] | Micron DDR3[22] |
| Rank Write B/W | 25.6(MB/s)[5] | 10.6(GB/s) |

## 3.2 Evaluation

To demonstrate the effectiveness of multi-level buffers, we consider a varying number of PRAM ranks. Also, for reference, we present corresponding data for a channel with four DRAM ranks.

**Experimental Setup.** To evaluate a PRAM memory channel with 64 ranks of PRAM devices, we use cycle-accurate M5[2] processor simulator augmented with modified DRAMSim2[28]. M5 is configured to execute the Alpha instruction set architecture in system-call emulation mode. DRAMSim2 is configured to simulate both DDR3-based DRAM and LPDDR2-N based PRAM with a 6.4(MB/s) write constraint [5]. Table 1 summarizes simulation parameters.

We use four different workloads. SPEC CPU2006 milc and mcf are used for their high cache miss and memory accesses per 1000 instructions [25]. In addition, two multi-programmed workloads are used. Multi-programmed I consists of astar,bzip2, gobmk, gcc, libquantum, milc, xalancbmk, and h264ref. This combination is used for its low locality and typical read-to-write ratio (1.5:1). Multi-programmed II consists of sjeng (4×) and bzip (4×), chosen for their high memory-level-parallelism.

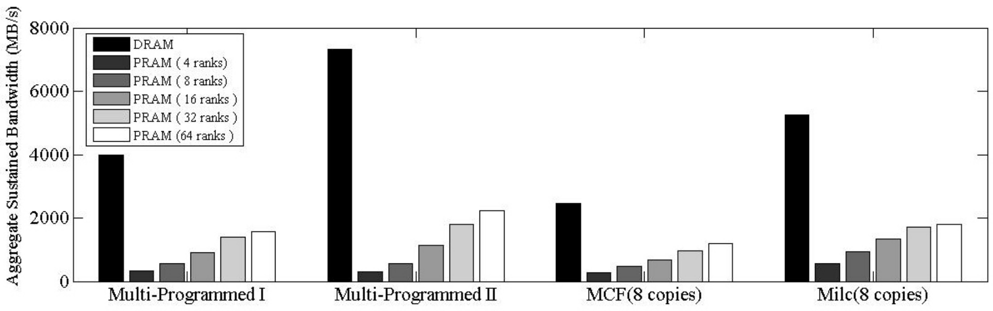

Figure 3: Sustained Bandwidth

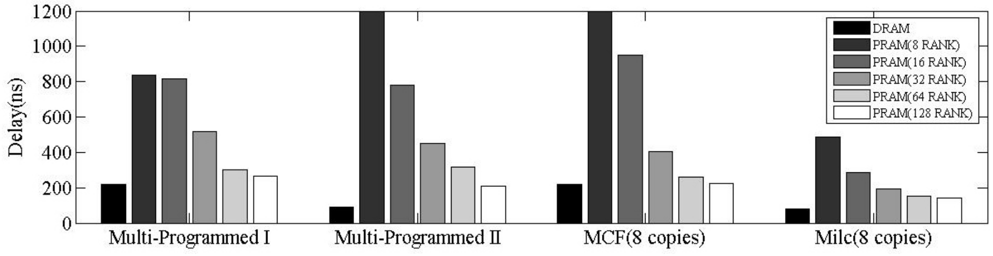

Figure 4: Average Read Latency

**Enhancing Bandwidth.** Applications can always sustain higher bandwidth on the DRAM subsystem. However, as the number of PRAM ranks increases from 4 to 64, Figure 3 shows monotonic increases in sustained PRAM bandwidth. Additional PRAM ranks exploit memory-level parallelism (MLP). Diminishing marginal returns reflect the limits of application MLP. We observe a small marginal benefit from doubling PRAM rank count from 32 to 64.

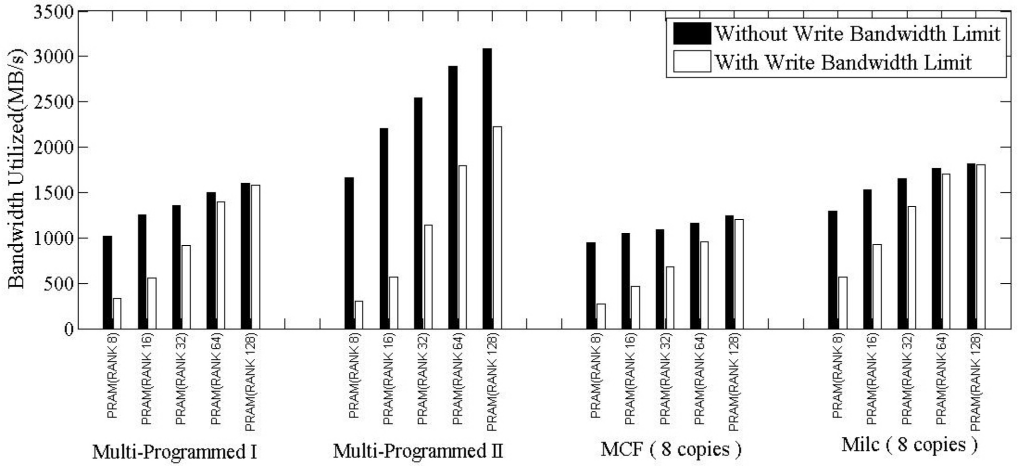

Figure 5: Write Bandwidth. Presented with and without constraints on PRAM programming

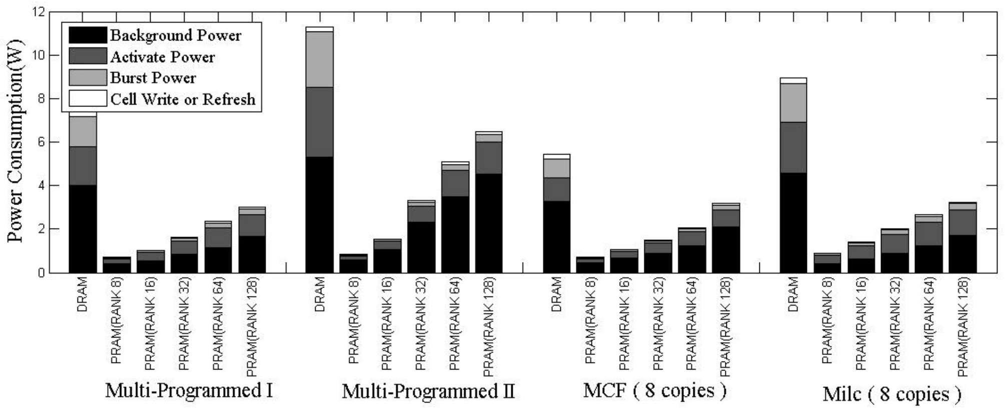

Figure 6: Power Breakdown

Unlike DRAM, phase change memory is characterized by asymmetric read and write performance. PRAM performance asymmetry is illustrated by Figures 4-5. Average PRAM read latency falls to approximately 200ns as the number of PRAM ranks increases, mirroring the bandwidth analysis. For most applications, average read delay is DRAM-competitive with 32 ranks. The exception is Multi-Programmed II, which is characterized by a low read-to-write ratio.

PRAM write performance is constrained by long-latency programming. A write injects current into a chalcogenide and a long current pulse is required to program the cell into a crystalline state. We consider x16 DRAM devices with a maximum 6.4(MB/s) write bandwidth [5]. Comparing against an idealized system without such a constraint, Figure 5 shows that rank-level parallelism exploits a large fraction of applications' write MLP with 64 ranks. The exception is Multi-Programmed II, which is write-intensive and benefits from an even larger number of PRAM ranks.

Thus, load-reducing buffers permit a PRAM architecture with a large number of ranks. And performance benefits from these ranks subject to their application-specific memory-level parallelism. As PRAM technology evolves, write bandwidth will continue to improve. While we consider devices with 6.4(MB/s) write bandwidth, recent prototypes indicate 40(MB/s) is possible [4], further improving PRAM subsystem performance.

**Optimizing Efficiency.** In addition to performance, an architecture should be power- and energy-efficient. Given four 2GB ranks, DRAM dissipates approximately 1.3(W/GB). Figure 6 indicates that low PRAM bandwidth leads to low PRAM power. First, PRAM background power benefits from using the LPDDR2-N interface, which operates at lower data rates to match low PRAM device bandwidth and, consequently, can forgo expensive delay-locked loops and on-die-termination. Second, PRAM writes may be energy-intensive but they incur long latencies, occur infrequently, and do not significantly impact power.

The marginal benefit in application bandwidth diminishes as the number of PRAM ranks increases and an efficiency-maximizing design exists. We consider power divided by utilized

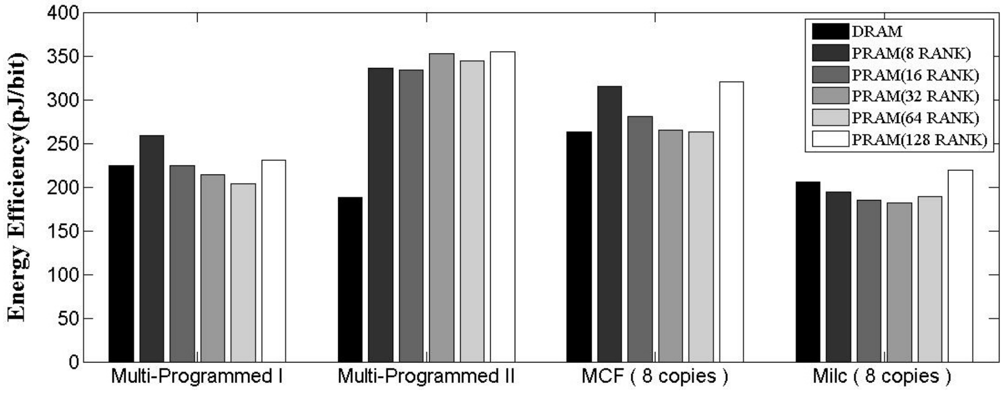

Figure 7: Energy Efficiency (pJ/bit = mW/Gbps)

bandwidth to estimate average energy per bit (pJ/bit = mW/Gbps). Figure 7 shows DRAM and PRAM exhibit comparable efficiency, primarily because PRAM sustains much lower data bandwidth while dissipating much less power. A system with 32 PRAM ranks is energy-efficient. Multi-Programmed II is an exception; bandwidth improves up to 64 PRAM ranks, justifying higher power costs.

## 4 HPC Checkpointing

A Massively Parallel Processing (MPP) system consists of many individual processor/memory nodes. MPP systems have high fault rates; any single processor fault induces system failure. For example, an ASCI Q supercomputer at Los Alamos National Laboratory has mean time to interrupt (MTTI) of less than 6.5 hours [27]. Future MPP systems have higher failure rates as processor counts increase. Failure rate is proportional to processor count [13]. Future MTTI may be less than an hour [13].

MPP systems use checkpoint/restart for fault tolerance, periodically saving state to persistent stores and restarting workloads from saved state after a failure. With large MTTI, checkpointing is infrequent and overheads are negligible. However, decreasing MTTI increases checkpoint frequency and overheads will be substantial. Checkpointing to hard disk (HDD) incurs a 19% overhead in a petaflop machine and may exceed 97% at 10 petaflops [11].

To reduce checkpoint overheads, Dong et al. first propose exploiting the speed of phase change memory for MPP checkpointing [11]. To provide bandwidth, this prior architecture relies on either x72 PRAM devices that are 6× wider than existing PRAM prototypes or 3D-stacked DRAM/PRAM with through-silicon-vias.

We present an alternative architecture that leverages existing technologies, such as fast point-to-point links, disintegrated memory controllers, and memory buffers. Our architecture does not require changes to PRAM devices. Rather we organize PRAMs to achieve the desired system bandwidth and checkpoint overheads.

### 4.1 PRAM/DRAM for Checkpointing

In hybrid checkpointing [11], the system periodically checkpoints to local storage (local checkpointing). With a longer period, local checkpoints are reflected to centralized I/O nodes (global checkpoint). When a node fails, it recovers from a local checkpoint. If any local checkpoint is lost, the node must instead recover from the global checkpoint.

The heterogeneous DRAM-PRAM architecture supports local checkpoints.[1] To be effective, the PRAM architecture should have sufficient bandwidth. To simplify the analysis and following Dong et al. [11], we assume DRAM and PRAM can realize their respective peak bandwidths when checkpointing. While unrealistic for general-purpose computation, this assumption holds for checkpointing, which exhibits streaming behavior and high spatial locality. Ideally, the PRAM subsystem's write bandwidth should match DRAM subsystem's read bandwidth.

[1]Although PRAM could also store global checkpoints, maintaining a large capacity PRAM system for these infrequent checkpoints would be inefficient.

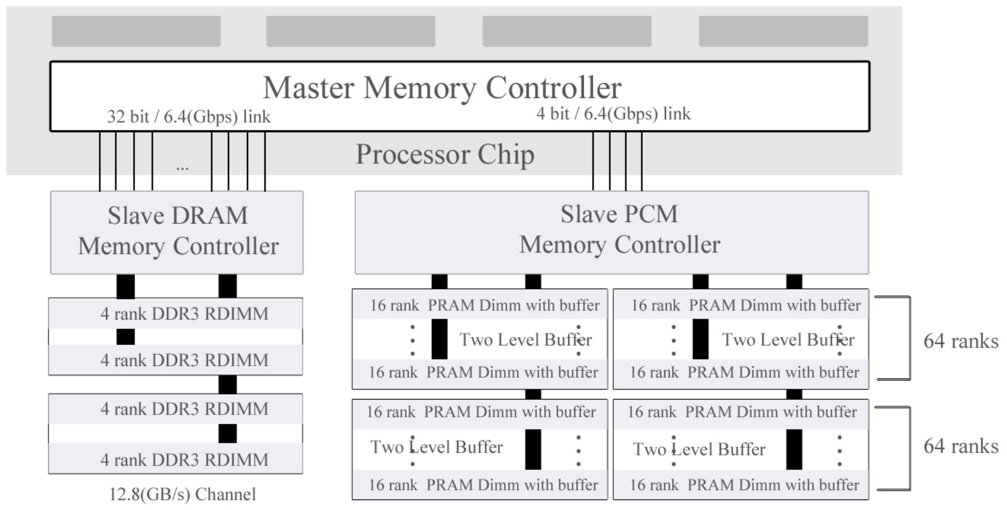

Figure 8: Configuration for PRAM/DRAM heterogeneous memory for checkpointing.

**Balancing Bandwidth.** Consider a heterogeneous memory architecture with DRAM and PRAM, as shown in Figure 8. Bandwidths should be balanced across master-slave point-to-point links, the DRAM subsystem, and the PRAM subsystem. First, observe that serial links are fast and are not a bottleneck. The 32bit serial links with 6.4(Gbps) per link supports an aggregate bandwidth of 25.6(GB/s). Second, in the DRAM subsystem with two DDR3-1666 channels, we have 25.6(GB/s) of DRAM bandwidth.

With multi-level buffers, we have a mechanism to enhance and balance PRAM bandwidth. The balance of DRAM read and PRAM write bandwidths determines checkpoint overheads. We architect the PRAM subsystem with LPDDR2-N-based x16 PRAM devices, each with 6.4(MB/s) of programming bandwidth [5]. With four devices per rank, a PRAM rank has 25.6(MB/s) write bandwidth.

As we increase the number of PRAM ranks, memory-level parallelism and peak bandwidth increases. We consider different numbers of PRAM ranks, expressing PRAM write bandwidth as a percentage of DRAM read bandwidth. For example, 128 PRAM ranks provide an aggregate write bandwidth of 3.3(GB/s), which is approximately 12.5% of DRAM's 25.6(GB/s) read bandwidth. And we architect the requisite number of ranks by determining the number of ranks that can occupy a channel, the number of buffers required, and the number of levels in a buffer hierarchy. An example configuration is shown in Figure 8.

DRAM capacity determines the amount of data that must be copied to PRAM during a checkpoint. For example, if a DDR3 rank has 2(GB) capacity, several channels might support 16 DRAM ranks and provide 32(GB) of capacity. To copy this data to PRAM for a local checkpoint, we consider PRAM write bandwidth: checkpoint time = $32(GB)/3.3(GB/s) = 10(s)$. Exhibiting high spatial locality and leveraging the system's sequential bandwidth, checkpoint data streams from the DRAM slave through the master to the PRAM slave. This bandwidth analysis drives an analytical evaluation of HPC checkpointing overheads.

## 4.2 Analytical Hybrid Checkpoint Model

Daly proposed an analytical cost model to estimate checkpointing overheads [8]. Dong et al. extend this model for hybrid checkpointing, which performs most checkpoints locally to persistent memory and a few checkpoints globally to a centralized I/O servers. Table 2 lists the parameters needed to calculate hybrid checkpoint overhead.

Table 2: Hybrid Checkpointing Parameters

| Parameter | Meaning |
|---|---|
| $T_s$ | Original computation time |
| $T_{total}$ | Total execution time |
| $\tau$ | Checkpoint interval |
| $p_L$ | Percentage of local checkpoint |
| $p_G$ | Percentage of global checkpoint |
| $\delta_L$ | Local checkpointing time |
| $\delta_G$ | Global checkpointing time |
| $\delta_{eq}$ | Equivalent checkpoint time |
| $R_L$ | Local recovery time |
| $R_G$ | Global recovery time |
| $R_{eq}$ | Equivalent recovery time |
| $q_L$ | Percentage of transient failure |
| $q_G$ | Percentage of permanent failure |
| $MTTF$ | Mean time to failure |

$$\begin{aligned} T_{total} &= T_S \\ &+ \frac{T_S}{\tau}(\delta_{eq}) \\ &+ \frac{T_{total}}{MTTF} \times \left(\frac{1}{2}(\tau + \delta_{eq}) + R_{eq}\right) \\ &+ \frac{T_{total}}{MTTF} \times q_G \times \frac{p_L}{2p_G}(\tau + \delta_L) \end{aligned}$$

**Model Background.** The model has several parts. The first part is the original computation time ($T_S$). Checkpointing incurs an additional overhead as the system performs periodic writes that require time ($T_S/\tau \times \delta_{eq}$). Since $\tau$ is the local checkpoint interval, this expression captures the number of local checkpoints. Each checkpoint incurs an average cost of $\delta_{eq}$, which accounts for the percentages of local versus global checkpoints ($p_L$ versus $P_G$) and the time required for each ($\delta_L$ versus $\delta_G$): $\delta_{eq} = p_L \times \delta_L + p_G \times \delta_G$.

When a failure occurs, recovery costs are incurred. The number of recoveries is estimated by total run time divided by mean time to failure ($MTTF$). The model assumes failures occur half-way through a compute interval ($\frac{1}{2}(\tau + \delta_{eq})$) and incur a recovery cost $R_{eq}$, which accounts for the percentages ($q_L$ and $q_G$) of failures recoverable by local and global checkpoints and the recovery times ($R_L$ and $R_G$): $R_{eq} = q_L \times R_L + q_G \times R_G$.

Finally, if a failure has to rely on the global recovery (likelihood is $q_G$), additional useful computation is lost and must be re-computed. On average, the number of local checkpoints between two global checkpoints is $\frac{p_L}{p_G}$. On average, failures occur half-way through this interval and the amount of time lost to wasted computation that must be recomputed is $\frac{p_L}{2p_G}$ multiplied by the length of the interval ($\tau + \delta_L$).

Total computation time is assumed to be 720(hr). Reboot time during recovery is assumed to be 0(s) for simplicity. Then, we can state that $R_{eq} = \delta_{eq}$. We assume $MTTF = 3(hr)$ and assume transient fault rate $q_L = 95\%$.

**Optimization.** Given a memory architecture that incurs particular local checkpointing costs ($\delta_L$), we optimize the equation to minimize checkpointing overhead by consider different ratios for local and global checkpoints ($p_L = 100 - p_G$) and the interval of those checkpoints $\tau$. The optimization is performed using numerical methods in Matlab. For example, 128 PRAM-rank topology being optimized is shown in Figure 9.

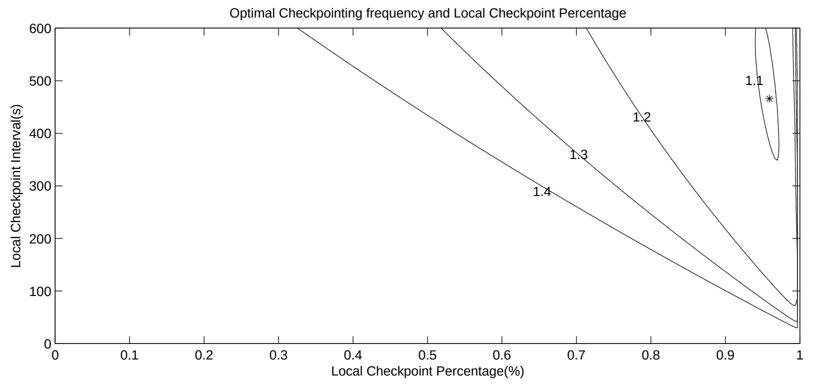

Figure 9: Optimizing checkpoint period (s) and the percentage of local versus global checkpoints(%).

## 4.3 Evaluation

Table 3: Baseline System Parameters [12]

| **Attribute** | **NCSA Blue Waters** |
|---|---|
| Processor | IBM Power 7 |
| Peak Performance | >=10 PF |
| Number of Cores per chip | 8 |
| Number of Processor Cores | 300,000 |
| Number of CPU chips | 37500 |
| Amount of memory | 1200(TB) |
| Memory per CPU socket | 32(GB) |
| Disk Transfer Rate | 4(TB/s) |
| MTTF (expected) | 3(hrs) |
| Transient Faults (expected) | 95(%) |

We consider a high-performance computing system expected to achieve 10 petaflops, as shown in Table 3. For this system, we apply a memory architecture with disintegrated memory controllers and multi-level buffers for a high-capacity, high-bandwidth PRAM subsystem. The heterogeneous architecture reduces checkpoint overhead, defined as $(T_0 + T_c)/T_0$ where $T_0$ is execution time without checkpointing and $T_c$ is time spent checkpointing.

We quantify these reductions with analytical models proposed by Daly [8], extended by Dong [11], and described in the subsection above. The model is parameterized to estimate overheads as a function of checkpoint frequency and the percentage of local versus global checkpoints. The time required for global checkpoints depends on memory capacity across all nodes and disk I/O bandwidth. In this system, the global checkpoint delay is $1200(TB)/4(TB/s) = 300(s)$.

The time required for local checkpoints depends on a single node's memory capacity and PRAM write bandwidth. Table 4 shows local checkpoint time as a function of the number of PRAM ranks. As the number of PRAM ranks increases, so does write bandwidth of the PRAM subsystem, measured as a percentage of DRAM read bandwidth. As bandwidth increases, local checkpoint time falls.

Given an analytical model of checkpoint overheads, we further optimize the checkpoint period and the percentage of these checkpoints that are local to minimize overhead for a given number of ranks. Given a particular number of ranks, the optimization determines the checkpoint period (s) and the percentage of local versus global checkpoints.

After optimizing checkpoint overhead, we compare checkpointing with hard disks against hybrid checkpointing with heterogeneous DRAM/PRAM. If checkpoints are always global and write to disk, the optimum checkpointing interval is 2545(s), incurring an overhead of 27.7%. Hybrid checkpoints rely on local writes to PRAM. And if 128 ranks provide write bandwidth that is 12.5% of DRAM read bandwidth, the local checkpoint delay is 10(s). This architecture checkpoints every 466(s) and 96% of these checkpoints are local to each node. The overhead is 9.8%.

Thus, hybrid checkpointing shifts the checkpointing delays to local nodes that benefit from high-bandwidth writes to persistent memory. Although hybrid checkpointing performs global

Table 4: Checkpoint (CP) Overheads for Varying PRAM Ranks. PRAM bandwidth expressed as a percentage of DRAM bandwidth. *Note that checkpoint period and local checkpoint percentage are degrees of freedom, optimized to minimize overhead.

| PRAM Ranks (#) | PRAM B/W (%) | Local CP $\delta$ (s) | CP Period* (s) | Local CP* (%) | CP Overhead (%) |
|---|---|---|---|---|---|
| 32 | 3 | 40 | 933 | 91.5 | 14.3 |
| 64 | 6 | 20 | 659 | 94.1 | 11.7 |
| 128 | 13 | 10 | 466 | 95.9 | 9.8 |
| 256 | 25 | 5 | 330 | 97.1 | 8.5 |
| 512 | 50 | 2.5 | 234 | 98.0 | 7.6 |
| 1024 | 100 | 1.25 | 166 | 98.6 | 6.9 |

writes to disk, such writes are rare. With 96% of the checkpoints local, one global checkpoint is performed for every 24 local ones.

## 5 Related Work

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 [24]. This work assumes three different memory technologies would use DDR3 protocol via the same controller. In contrast, we architect disintegrated master-slave controllers.

Many heterogeneous memory systems use a small DRAM as a cache for a larger PRAM-based main memory [3, 9, 23, 26]. Caches mitigate the low PRAM bandwidth and enhances PRAM endurance. Orthogonal to our work, these caching and data management strategies could be implemented by the master in a heterogeneous architecture with discrete memory controllers.

In addition to quantitative performance differences, heterogeneous memories introduce qualitative capability differences. Coburn et al. and Volos et al. separately exploit heterogeneous non-volatile memory for persistent data structures by providing programmer-exposed primitives and abstractions [6, 30]. Our hardware supports these software mechanisms by assigning non-volatile memory to a particular range of addresses.

Condit et al. present a new file system that improves data safety and consistency with phase change memory [7]. Dong et al. propose stacking PRAM on DRAM for high-bandwidth HPC checkpointing [11, 10]. Using their analytical model for checkpointing overheads, we present an alternative architecture with discrete memory controllers and load-reducing buffers. Our approach is extensible with respect to the number of heterogeneous technology protocols as well as bandwidth and capacity.

## 6 Conclusion

We present a new architecture for heterogeneous memory controllers with an integrated master that forwards memory requests to discrete slaves, each of which implement heterogeneous protocols for different technologies. To enhance capacity and bandwidth, especially for emerging phase change memory, we use load-reducing buffers to construct a hierarchical channel architecture. With modest power and latency overheads, the system provides capacity and bandwidth that benefits a variety of applications such as HPC checkpointing.

## References

[1] AMD. ACP-The truth about power consumption starts here. Technical Report AMD-43761C, 2009.

[2] N. Binkert et al. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39:1–7, Aug. 2011.

[3] A. Bivens et al. Architectural design for next generation heterogeneous memory systems. In *International Memory Workshop (IMW)*, 2010.

[4] Y. Choi et al. A 20nm 1.8V 8Gb PRAM with 40MB/s program bandwidth. In *ISSCC*, 2012.

[5] H. Chung et al. A 58nm 1.8V 1Gb PRAM with 6.4MB/s program BW. In *ISSCC*, 2011.

[6] J. Coburn et al. NV-Heaps: Making persistent objects fast and safe with next-generation, non-volatile memories. In *ASPLOS*, 2011.

[7] J. Condit et al. Better I/O through byte-addressable, persistent memory. In *SOSP*, 2009.

[8] J. T. Daly. A higher order estimate of the optimum checkpoint interval for restart dumps. *Future Gener. Comput. Syst.*, 22, February 2006.

[9] G. Dhiman, R. Ayoub, and T. Rosing. PDRAM: A hybrid PRAM and DRAM main memory system. In *DAC*, 2009.

[10] X. Dong, N. Muralimanohar, N. Jouppi, R. Kaufmann, and Y. Xie. Leveraging 3D PCRAM technologies to reduce checkpoint overhead for future exascale systems. In *SC*, 2009.

[11] X. Dong, Y. Xie, N. Muralimanohar, and N. Jouppi. Hybrid checkpointing using emerging nonvolatile memories for future exascale systems. *ACM Trans. Archit. Code Optim.*, 8:6:1–6:29, June 2011.

[12] T. Dunning, B. Kramer, M. Snir, B. Gropp, and W. Hwu. Blue Waters: Leading the way in sustained petascale computing. Presentation, 2011.

[13] G. Gibson, B. Schroeder, and J. Digney. Failure tolerance in petascale computers. *CT-Watch Quarterly*, 3:4–10, 2007.

[14] Inphi. Basics of LRDIMM. `http://www.edn.com/article/519386-Basics_of_LRDIMM.php`, 2011.

[15] Intel. Intel 7500 scalable memory buffer datasheet, 2011.

[16] JEDEC. Low power double data rate 2 (LPDDR2), 2011.

[17] 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.

[18] R. Kalla, B. Sinharoy, W. Starke, and M. Floyd. Power7: IBM's next-generation server processor. *IEEE Micro*, 30(2), 2010.

[19] C. Kozyrakis. Memory management beyond free(). Keynote: International Symposium on Memory Management, 2011.

[20] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger. Architecting phase change memory as a scalable dram alternative. In *ISCA*, 2009.

[21] Micron. Calculating memory system power for DDR3. Technical Note TN-41-01, 2007.

[22] Micron. Ddr3 sdram datasheet. Technical Report MT41J1G4, 2009.

[23] H. Park, S. Yoo, and S. Lee. Power management of hybrid DRAM/PRAM-based main memory. In *DAC*, 2011.

[24] S. Phadke and S. Narayanasamy. MLP aware heterogeneous memory system. In *DATE*, 2011.

[25] A. Phansalkar, A. Joshi, and L. John. Analysis of redundancy and application blanace in the SPEC CPU2006 benchmark suite. In *ISCA*, 2009.

[26] M. K. Qureshi, V. Srinivasan, and J. A. Rivers. Scalable high performance main memory system using phase-change memory technology. In *ISCA*, 2009.

[27] D. Reed. High-end computing: The challenge of scale. Presentation: Director's Colloquium at Los Alamos National Laboratory, 2004.

[28] P. Rosenfeld, E. Cooper-Balis, and B. Jacob. Dramsim2: A cycle accurate memory system simulator. *IEEE Computer Architecture Letters*, pages 16–19, 2011.

[29] Samsung. Samsung DDR3 LRDIMM. `http://www.samsung.com/global/business/semiconductor/support/brochures/downloads/memory/samsung_LRDIMM.pdf`, 2010.

[30] H. Volos, A. J. Tack, and M. Swift. Mnemosyne: Lightweight persistent memory. In *ASPLOS*, 2011.

[31] R. Williams, T. Sze, D. Huang, S. Pannala, and C. Fang. Server memory road map. Presentation: Server Memory Forum, 2011.

[32] P. Zhou, B. Zhao, J. Yang, and Y. Zhang. A durable and energy efficient main memory using phase change memory technology. In *ISCA*, 2009.