# There and Back Again: Optimizing the Interconnect in Networks of Memory Cubes

Matthew Poremba* Itir Akgun*† Jieming Yin* Onur Kayiran*
Yuan Xie*† Gabriel H. Loh*
Advanced Micro Devices, Inc.*; University of California, Santa Barbara†
{matthew.poremba,jieming.yin,onur.kayiran,gabriel.loh}@amd.com,{iakgun,yuanxie}@ece.ucsb.edu

## ABSTRACT

High-performance computing, enterprise, and datacenter servers are driving demands for higher total memory capacity as well as memory performance. Memory "cubes" with high per-package capacity (from 3D integration) along with high-speed point-to-point interconnects provide a scalable memory system architecture with the potential to deliver both capacity and performance. Multiple such cubes connected together can form a "Memory Network" (MN), but the design space for such MNs is quite vast, including multiple topology types and multiple memory technologies per memory cube.

In this work, we first analyze several MN topologies with different mixes of memory package technologies to understand the key tradeoffs and bottlenecks for such systems. We find that most of a MN's performance challenges arise from the interconnection network that binds the memory cubes together. In particular, arbitration schemes used to route through MNs, ratio of NVM to DRAM, and specific topologies used have dramatic impact on performance and energy results. Our initial analysis indicates that introducing non-volatile memory to the MN presents a unique tradeoff between memory array latency and network latency. We observe that placing NVM cubes in a specific order in the MN improves performance by reducing the network size/diameter up to a certain NVM to DRAM ratio. Novel MN topologies and arbitration schemes also provide performance and energy deltas by reducing the hop count of requests and response in the MN. Based on our analyses, we introduce three techniques to address MN latency issues: (1) Distance-based arbitration scheme to improve queuing latencies throughout the network, (2) skip-list topology, derived from the classic data structure, to improve network latency and link usage, and (3) the MetaCube, a denser memory cube that leverages advanced packaging technologies to improve latency by reducing MN size.

## CCS CONCEPTS

• **Hardware** → **Emerging architectures**; *Memory and dense storage*; Emerging interfaces;

## KEYWORDS

high-performance computing, memory cube, memory network, non-volatile memory

**ACM Reference format:**
Matthew Poremba* Itir Akgun*† Jieming Yin* Onur Kayiran* Yuan Xie*† Gabriel H. Loh* Advanced Micro Devices, Inc.*; University of California, Santa Barbara†. 2017. There and Back Again: Optimizing the Interconnect in Networks of Memory Cubes. In *Proceedings of ISCA '17, Toronto, ON, Canada, June 24-28, 2017,* 13 pages.
https://doi.org/10.1145/3079856.3080251

## 1 INTRODUCTION

Computing trends in multiple areas are placing increasing demands on the memory systems of modern servers, supercomputers, and even high-end workstations. Greater total memory capacity is required to solve huge scientific problems, tackle massive data analytics challenges, and host larger numbers of virtualized servers. For conventional memory systems (e.g., DDR4), increasing memory capacity comes with a performance penalty. A typical server processor package has a limited number of memory channels. If a single dual-inline memory module (DIMM) is placed on a channel, the bus speed can typically be operated at its maximum speed. However, as the number of DIMMs per channel increases, the electrical loading often requires the bus to be operated at a lower rate, trading memory bandwidth of the slower bus for greater memory capacity of extra DIMMs. Furthermore, the ability to increase capacity is typically limited to a few (e.g., three) DIMMs per channel.

The other approach to increase memory capacity is to increase the number of memory channels per processor package. However, this is very expensive in terms of the number of pins required and the subsequent impact on package design. The DDR4 interface requires 288 pins per channel; e.g., a typical four-channel server processor would need 1,152 pins for memory interfaces alone. We do not further explore increasing memory capacity by adding more channels in this work.

One recent alternative approach is to combine 3D-stacking technology for higher per-package capacity with narrower high-speed serial links, internal switches, and an abstracted interface (as opposed to low-level DRAM commands). The primary demonstrator of this is embodied in Micron's Hybrid Memory Cube (HMC) [33], but the concept is being espoused by others [36, 37].

Current HMC capacity is 2-4GB per cube [33], although other 3D-stacked memory technologies, such as JEDEC's HBM2, will provide up to 8GB per DRAM stack [39]. Furthermore, the memory capacity and bandwidth are expected to double with HBM Gen3 [7], so there is no reason that a memory cube could not support similar capacities. By chaining (or arranging in other topologies) multiple memory cubes, system capacity can be scaled while maintaining high bandwidth through the use of high-speed point-to-point links (i.e., unlike DDR, link speed does not fall off with more packages). The tradeoff is the interconnect latency to reach a memory cube can become non-trivial, especially if the destination memory cube requires multiple hops to reach.

In this paper, we start off with a performance analysis of a variety of different topologies of memory networks (MNs). From this, we identify several bottlenecks in building large, terabyte-scale memory systems. There is no single Achilles Heel for MNs, but the performance challenges come from a combination of the latency of the underlying memory technology, the multi-hop interconnect latency, and queuing/arbitration issues in the MN. Based on these observations, we recommend several improvements to MNs to increase performance for large-capacity memory systems. These include a new MN topology inspired from classic "skip list" data structures, a distributed arbitration mechanism to improve overall MN throughput, and the mixing in of non-volatile memory technologies to achieve different tradeoffs between interconnect latency (MN size/diameter) and memory array latency. We also introduce a "cube-of-cubes" concept that can further improve performance, especially for systems with very large capacities.

## 2 BACKGROUND

In this section, we briefly discuss the limitations of the conventional DDR interfaces and introduce recent memory technologies that we use to form MNs, particularly, memory cubes and non-volatile memories.

### 2.1 Limitations of DDR

As discussed in the introduction, bus-based memory interfaces such as DDR4 typically take a performance hit as memory capacity is increased. As more DIMMs are added to the bus, the increase in electrical loading makes it harder to run the interface at the highest speeds. The exact supported bus speeds for a given number of DIMMs per channel (DPC) varies by manufacturer and server part. Table 1 shows example maximum bus speeds for DDR3 and DDR4 memories given different numbers of DPC. For DDR3, to operate at the higher bus speed of 1333 MHz, only one DPC can be populated. To achieve higher capacity with two DPC, the bus speed drops to 1066 MHz, and three DPC goes all the way down to 800 MHz.[1] DDR4 has some improvements (especially if registered (RDIMM) or load-reduced DIMMs are used), but the same general phenomena apply. When going from one DPC to two, there is no drop-off in bus speed, but adding the third DPC still incurs the performance penalty. Attempts to support even more DPC to further increase capacity exacerbate the electrical challenges of high-speed signaling due to the additional impedance on the bus. For this reason in many servers, three DPC is the limit supported. These examples illustrate the fundamental tradeoff between memory capacity and memory performance in conventional bus-based, multi-drop memory interfaces.

[1]There are variations where multi-rank DIMMs can cause the bus speed to drop compared to a single-rank DIMM even for the same number of DPC, but such details are not key to the current discussion.

**Table 1: Maximum memory interface speeds given different numbers of DIMMs per channel for DDR3 [10] and DDR4 [15].**

| Number of DPC | DDR3 | DDR4 |
|---|---|---|
| 1 | 1333 MHz | 2133 MHz |
| 2 | 1066 MHz | 2133 MHz |
| 3 | 800 MHz | 1866 MHz |

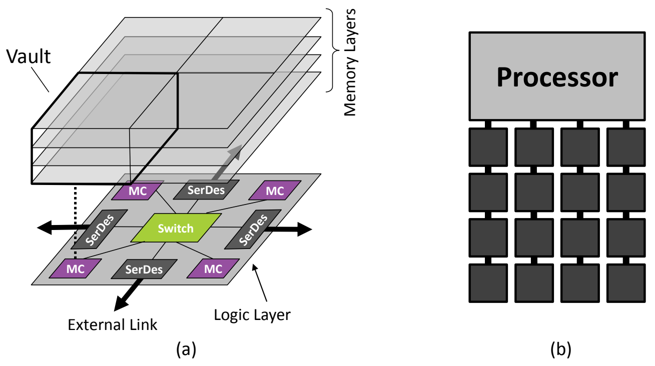

**Figure 1: (a) Conceptual view of a Hybrid Memory Cube with a logic layer and four memory layers, and (b) an example processor with multiple HMCs arranged as four parallel chains.**

### 2.2 Memory Cubes

The maturity of 3D die-stacking technology along with advancements in short-range, high-speed signaling, have enabled a new memory package organization. The pioneering example of this approach is embodied in Micron's Hybrid Memory Cube (HMC) [20, 21]. Fig. 1(a) shows a conceptual depiction of a memory cube. 3D stacking is used to integrate multiple layers of DRAM on top of a base logic die. The logic die implements an array of memory controllers tightly coupled to the vertical DRAM channels ("vaults") through arrays of through-silicon vias (TSVs), which has additional performance and energy benefits from having the controllers physically close to the DRAM with a fixed amount of electrical loading (a single controller-to-vault interface is represented by the vertical dashed line in the figure). The logic die also implements high-speed serial links to the outside of the package. These links can be used to connect to a host processor or to other memory cubes. A switch on the logic die connects the external links to the internal memory controllers. HMC's logic die also implements other functionality such as built-in self-test and error correction, but we do not explore these additional types of features in this work.

The external links of a memory cube can be operated at very high speeds in part due to the point-to-point nature of the memory organization. Fig. 1(b) shows a processor connected to four chains with four memory cubes per chain. Each link starts at one package and ends a short distance away at the next package; this is in contrast to a DDR bus that traverses a greater distance on the motherboard and may have a variable number of DIMMs. HMC 1.0 offered links speeds up to 15 Gbps, for a peak aggregate bandwidth of 240 GB/s [20]. HMC 2.0 (short-range version) supports link speeds up to 30 Gbps (peak aggregate bandwidth of 320 GB/s/link) [21]. In contrast, the fastest current DDR4 bus speeds are only 1.6 Gbps (25.6 GB/s/channel). Furthermore, the DDR4 interface requires 288 pins per channel, while HMC 2.0 only needs 66 pins per link; i.e., for the same pin cost on the processor package, one could have over four times the number of HMC 2.0 links (this is a simplistic comparison as there are other costs apart from pins).

Another advantage of the memory cube approach is that the package-to-package links make use of a much more abstract interface than DDR [36]. DDR requires the host processor's memory controller to send a sequence of explicit, low-level, carefully-timed commands (e.g., precharge, row activation, column read). With an abstract interface, an asynchronous protocol is instead employed, where the host processor can, for example, issue a "read" command to the memory cube, which then internally can decide how (and with what timing) to read the data from the DRAM. Whenever the data access has completed, the memory cube's logic die formulates a response packet to be sent back to the host processor. Such an abstracted interface decouples the semantic actions (e.g., read) from the low-level device behaviors (e.g., row activation).

**Baseline Memory Cube:** We briefly describe our baseline assumptions for the organization of the memory cubes assumed throughout the rest of this paper.[2] The logic die at the bottom of the stack contains the SerDes (serialization-deserialization) interfaces for the external high-speed links. The SerDes are coupled to a switch that either forwards requests out of the appropriate external link, or to one of the internal memory controllers. The memory stacked on top of the logic die provides multiple channels for independent and concurrent accessing of the memory resources. We do not assume any specific memory layout; vertical "vaults" like in HMC could be used, or each layer could implement one or more independent channels as in HBM. The memory controller receives a request, and then handles any further processing (e.g., issuing of device-level sub-commands) required to complete the request.

While memory cubes could support additional functionality, this work does not consider any further capabilities beyond routing requests and performing reads and writes of memory (along with any basic "maintenance" operations required for proper operation, e.g., refresh). These could include but are not limited to built-in self-test, ECC, repair, thermal sensing and management, or processing in-memory features. The switch within the logic layer could be a single monolithic multi-ported switch, or it could consist of multiple switches or routers arranged as a small network on chip.

## 2.3 Memory Networks

Industry efforts from HP [1] and IBM [32] demonstrate the trend towards interconnecting memory modules to enable efficient main memory expansion. In academia, several key prior works have already started the exploration of Memory Networks (MNs). Kim et al. [24] introduced the term Memory-Centric Network (MCN) and then later shortened it to just Memory Network (MN) [26]. We follow the more succinct MN terminology here.

The past works have considered a variety of computing scenarios including multiple CPU packages and CPUs with multiple discrete GPUs. Examples of these are shown in Fig. 2(a) and (b). A key assumption in these past systems is that all memory cubes are connected to all other memory cubes. This may be appropriate for the multi-GPU systems where the MN not only provides the memory

[2]We use the term "Memory Cube" instead of Hybrid Memory Cube or HMC, because HMC is a definition for specific family of memory cubes. As this work considers future cube organizations, the more general term Memory Cube was deemed to be more appropriate (as well as to not be misleading about HMC).

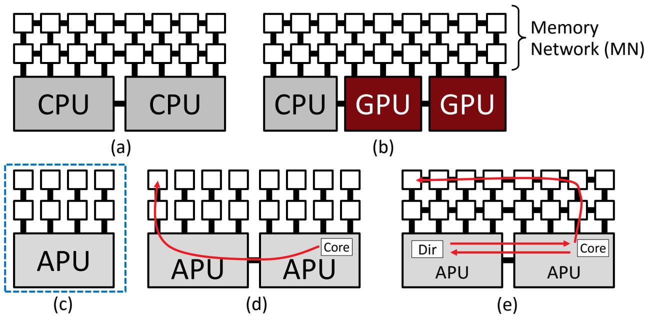

**Figure 2: Memory Networks (MNs) connecting (a) multiple CPUs and (b) a CPU with multiple GPUs. Disjoint per-port MNs for (c) one APU and (d) two APUs. (e) Example showing the ineffectiveness of fully-connected MNs in a cache-coherent multi-package system.**

storage, but also the datapath for the disparate GPU units to access all of the shared memory.

In this work, our baseline system consists of a high-performance heterogeneous processor consisting of CPU and GPU resources integrated in a single chip (sometimes referred to as an accelerated processing unit or APU). The APU supports multiple memory ports, but we make a key assumption that simplifies the MNs. In particular, as is typically done for systems today, the physical memory address space is interleaved across the APU's memory ports (this applies for CPU-only configurations as well). As a result, each memory port serves a *disjoint* subset of the memory space, which means that there is no need for memory cubes from one port to be connected to memory cubes on another port.[3]

Fig. 2(c) and (d) shows examples of such MN organizations for one and two APU packages, respectively. Note that in particular for the dual-package configuration in Fig. 2(d), requests from the right APU package destined for the left set of memory cubes must first route to the left APU package. At first glance, it would seem desirable to route such requests directly to the left memory cubes through a MN connecting all cubes, which would leverage the greater interconnect bandwidth of the MN and possibly reduce the end-to-end request latency. However in a cache-coherent system, requests would likely first need to route to the left APU anyway to consult cache coherence structures (e.g., directories); once a request has been routed to the left APU for coherence reasons, there is no point in sending the request *back* to the right APU only to re-route through the whole MN as shown in Fig. 2(e). As a result, we focus on MN organizations similar to those shown (highlighted with a dashed box) in Fig. 2(c).

## 2.4 Non-Volatile Memories

Various emerging memory technologies (e.g., phase-change memory, STT-MRAM, memristors, 3D-XPoint) are gaining traction to be used as replacement for, or together with existing technologies, in caches and main memories. This trend is primarily due to their advantages such as non-volatility, density, and decoupled sensing and

[3]The exception is to support reliability schemes where a hardware failure at one memory port can be compensated for by routing requests around through another port. As this work is not focused on RAS issues and our proposals can be generalized to include some of this type of redundancy, we maintain this simplifying assumption of disjoint per-port address spaces through the rest of this paper.

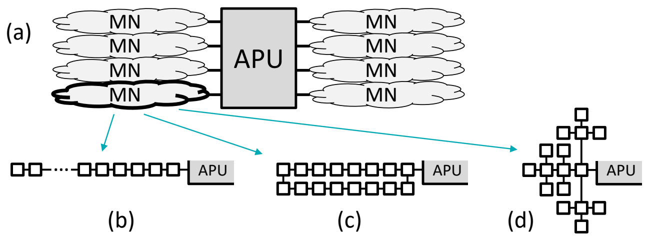

**Figure 3: (a) APU with eight memory ports, each with a disjoint MN. The MN topologies evaluated in this section include (b) chain, (c) ring, and (d) tree topologies.**

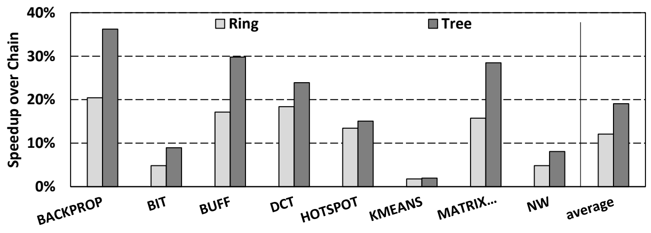

**Figure 4: Speedup comparison of DRAM memory networks normalized to a chain topology.**

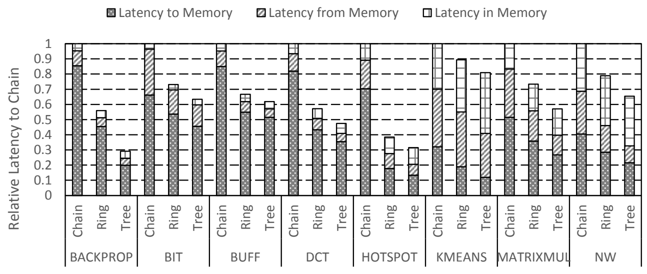

**Figure 5: Breakdown of memory request latency in DRAM memory networks normalized to chain topology.**

buffering. DRAM can entirely be replaced with NVM [27, 28], or can be reorganized in a multi-level hierarchy with different memory technologies. In this paper, we consider mixing DRAM and NVM memory cubes within the same MN. Such an organization requires management of data migration across different technologies/levels of memory, and metadata management. Prior works have attempted to address these issues [16, 30, 31, 41]; in this work we rely on the existence of appropriate heterogeneous management mechanisms (we do not try to re-invent anything on this particular topic as that is not the focus of the current paper).

## 3 BASELINE MN PERFORMANCE

We begin with a performance analysis and deconstruction of a few simple MN systems. Fig. 3(a) shows our baseline APU system. The APU has eight memory ports, each connected to an independent MN serving a disjoint slice of the global physical memory address space, as discussed in Section 2. The full details of the APU micro-architecture and other evaluation details can be found in Section 5. Connected to each of the APU's memory ports is a set of memory cubes interconnected in identical topologies. We study a system with a total memory capacity of 2TB, where each memory cube has a 16GB capacity (technology assumptions are also explained in Section 5). This leads to a total count of 128 memory cubes for the entire system, or 16 cubes per memory port. In terms of memory management, we make a conservative assumption that memory requests are uniformly interleaved based on their *addresses*. This means that for a system with 50% capacity from NVM, half of the memory requests will go to the NVM cubes. Although this work does not focus on the management of heterogeneous memory, prior work [17, 23, 30, 31, 41] exemplify how one can make use of the underlying MN more effectively.

We evaluate three different MN topologies. The first is a basic chain of memory cubes, shown in Fig. 3(b). This baseline minimizes the number of ports required per memory cube and has similarities with buffered DRAM topologies [18], but also suffers from large hop counts to reach the further memory cubes in the chain. The second is a ring topology, shown in Fig. 3(c), that halves the average hop count compared to the chain as requests can take the shorter of the upper or lower branches of the ring. The last topology is a tree, shown in Fig. 3(d), which in theory makes the most effective use of the memory cube's multiple external links. This results in a MN that has a worst-case hop count that increases only logarithmically with the number of memory cubes in the network. We do not consider a mesh in this work as the average hop count is larger than a tree no matter which memory cube is connected to the host, and for the reasons discussed in Section 2.3. Furthermore, we model HMC-like memory packages with 4 ports per package and only create networks utilizing the routers on the packages rather than adding specialized intermediate routing devices to the PCB. This means that high radix networks and networks requiring intermediate routers with no external link, such as flattened butterflies, distributed networks, all-to-all, etc. [24] are not possible.

### 3.1 Speedup Results

As a first cut, we compare a simple speedup of a MN consisting of all-DRAM cubes. Fig. 4 compares the speedup of the topologies normalized to the chain, which always provides the lowest performance. As expected, the topology with the fewest hops (i.e., tree) provides the best performance.

To further reveal potential bottlenecks in the topologies, we look at the breakdown of network latencies in both directions (to and from the destination cube) compared to the core access latency of the memory array.

### 3.2 Latency Breakdown

Fig. 5 shows the breakdown of latency to memory, latency in memory (i.e., memory array access), and latency back from the memory cube. We make three key observations revealing the potential bottlenecks that we later address in Section 4.

First, in most workloads, the latency spent in the network is significantly higher than latency of the memory core. This is especially true during periods of high network load. The discrepancy between to and from memory latencies occurs due to the network's single link which by default prioritizes responses (i.e., read data or write acknowledgment) over requests in order to prevent deadlocks from older responses being blocked by newer requests. This causes higher queuing latency at the outgoing memory port as requests are backed up waiting for responses to free up buffer slots. Overall, we observe that the topologies with fewer hops greatly reduce the network latency while the latency in memory remains relatively constant.

Second, differences between latency breakdowns across workloads is caused by the ratio of reads to writes in workload. We assume read responses and write requests (i.e., packets with data) are 5 times larger than control packets (i.e., read request and write response). For example, the workloads KMEANS, MATRIXMUL, and NW all have at least two reads for every one write. The BACKPROP workload has significantly more writes than reads, and the remaining workloads have nearly identical numbers of read and write requests. The NW workload also has the lowest network load of all the workloads plotted, and therefore has the highest percentage of latency spent performing the memory array access due to less contention in the network.

Last, to gain additional insights into where the cycles are being spent, we analyzed the waiting times observed in the various structures in the per-cube routers. In particular, we observed that the queuing latencies for the router input-ports were highly unbalanced, with the cubes closer to the processor showing more problems. This is because the default router implementation uses a locally-fair round-robin arbitration scheme that picks from each input queue in a uniform manner. However, four of the input queues come from the cube's local memory vaults, while the remaining input queues serve return traffic from other cubes. As an extreme example, the chain topology would pick the local input queues four out of five times (80% service) while picking the port connecting to the rest of the chain only one out of five times (20% service). Such unfairness in multi-stage arbitration has been previously observed for a two-stage 3D router [22], but with up to *n* stages in a MN, the effects are that much worse. Hence, in Section 4.1 we propose a globally fair arbitration technique to overcome the unbalanced queuing latencies.

## 3.3 Heterogeneous Mix of Cubes

The somewhat unsurprising observation that having network latency scale with the number of hops leads us to the conclusion that introducing higher density memory cubes can allow us to reduce the number of hops to decrease network latency. Using NVMs in memory cubes can provide this density benefit at the cost of higher latency. However, because of the magnitude of the network latency, the increase in access latency may be worthwhile to save on network latency while still reducing the overall round-trip latency. The heterogeneity in the memory network introduces non-uniformity in memory access, in terms of both topology and memory technology. Therefore approaches addressing non-uniform memory access (NUMA) systems can also be considered for MNs.

In the limit, given *n* memory cubes and a bound on the number of ports/links per cube (as well as motherboard routability concerns), topology selection can only go so far to reduce the number of hops through the MN interconnect. Another approach is to attempt to reduce *n* directly. To do this, we take advantage of emerging non-volatile memory technologies that provide greater storage density compared to conventional DRAM. For a memory technology such as phase-change memory (which is what we assume in our evaluations), we project that a memory cube can provide 4× as much capacity (e.g., 64GB per NVM cube compared to our baseline 16GB per DRAM cube). If all cubes are replaced by NVM, then the MN size

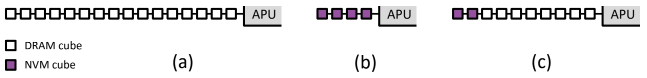

**Figure 6: Example MN topologies using (a) only DRAM memory cubes, (b) only NVM memory cubes with 4× the capacity per cube, and (c) a mix of both DRAM and NVM where each *type* provides half of the MN's total capacity. The figure only shows a single memory port, configuration is repeated across all ports.**

can be reduced to only $\frac{n}{4}$ cubes, with a corresponding reduction in MN hop counts. However, this is not free, because the latency and energy to read, and especially to write, for NVM is worse than of DRAM.

Instead of implementing a MN with only DRAM (minimizing memory array latency, shown in Fig. 6(a)) or with only NVM (minimizing interconnect latency, shown in Fig. 6(b)), we propose to build the MN with a mix of memory technologies. By varying what fraction of a MN's capacity is provided by DRAM versus NVM, we can vary how much of a request's latency (on average) comes from interconnect versus memory technology. Fig. 6(c) shows a MN where half of its capacity is provided by DRAM and the other half from NVM. This results in two NVM cubes and eight DRAM cubes, for a total MN size of *n*=10 cubes (compared to *n*=16 for the all-DRAM case). The examples in Fig. 6 show chain topologies simply for ease of illustration; other topologies can be constructed.

We performed a similar performance analysis using different ratios of NVM to DRAM. Throughout this work, the different ratios of NVM to DRAM are labeled by the percentage of DRAM in the MN by capacity. For any MN where multiple technologies exist, it is possible to change the locations of the memory cubes in the network. In other words, a MN with both NVM and DRAM can choose to place the NVM further away from or closer to the system port. These are differentiated using the suffixes -L (last) and -F (first), respectively. This location refers to the position of the NVM cubes. The topologies are differentiated using their first letter as the suffix. For example, 50%-C (NVM-L) is a MN designed with a chain topology with 50% DRAM and 50% NVM, where the NVM cubes are placed further away from the processor. A 50%-C (NVM-L) configuration is depicted in Fig. 6(c).

Fig. 7 shows the speedup results for the tree topology only. These are again normalized to the performance of a 100% Chain. The key takeaways from this figure are that it is beneficial to use some amount of NVM in the MN. There is no clear best topology in this case, however, the 50%-T (NVM-L) topology performs the best on average. The 0%-Tree varies highly with the workload. Workloads with the lowest network contention tend to show degradation, for example NW. This happens since the latency spent in network is not as prominent as in other workloads, where the dramatic reduction in hop count begins to reduce latency more than the NVM cubes increase latency.

The overall latency breakdown consists of very similar results to the all-DRAM case. In particular, there is very unbalanced request and response latency, whereby most workloads exhibit behavior where the request path is significantly longer than the response path. Additionally, workloads with higher read to write ratios typically

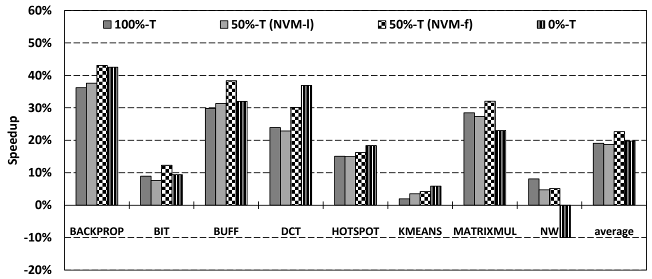

**Figure 7: Performance analysis of a Tree-based topology with different ratios of DRAM to NVM. 100% implies all DRAM while 0% implies all NVM. NVM-L places NVM cubes last (further away from the processor) while NVM-F places NVM cubes first (closer to the processor).**

have more balanced latency breakdowns. Due to this similarity, we focus on network latency optimizations over memory core latency.

## 4 ADDRESSING MN LATENCY ISSUES

The analysis from the previous section showed that various factors of the MN interconnection are responsible for the majority of a request's end-to-end latency. In particular, low hop count topologies such as tree topology performed significantly better than a ring or chain topology. Additionally, we observed a large increase in latency within the network due to arbitration techniques and packet prioritization.

In this section, we first introduce a distance based arbitration technique which reduces unbalanced input port selection by attempting to service older requests first. We then propose two topologies which attempt to exceed the performance of a tree topology. The first topology exploits the fact that memory traffic is kept coherent beyond the system port and uses this to provide more paths based on request type. The second is a cluster-like topology which leverages die-stacking further in order to reduce the hop count beyond that of a tree topology.

### 4.1 Fair, Distributed Arbitration

Section 3 showed how a locally fair round-robin arbitration on a memory cube's router can lead to global unfairness by introducing significant queuing delays for the ports connected to more distant cubes, also referred to as the "parking lot problem" [40]. If the router was made aware of how many requests were pending not only at a particular input port, but also downstream at any additional memory cubes that have to eventually flow back through this port, then the router arbitration could use a weighted round-robin approach where the probability of picking a port is weighted by the number of downstream requests that are trying to make their way back to the host. While this approach would work well, it requires routers to maintain global knowledge of all other downstream memory cubes, which would be difficult to implement in practice.

Another alternative is to use age-based prioritization. If the router arbiter knows which requests are the oldest, then no request will be starved for very long. In particular, if all requests are injected into the MN at approximately the same time (subject to the injection bandwidth limit on the first MN link), then having each router favor the oldest requests will tend to cause all of these requests to be returned to the host ahead of any later requests. This approach also performs well to alleviate high-contention scenarios, but unfortunately it also has a significant implementation challenge. In particular, each message would have to have some form of timestamp embedded in it. However, typical flit header formats do not provide very many (if any) unused bits that could be used to store a timestamp.

We make the observation that in general, requests that are destined for cubes that are farther away (larger hop count) will have longer end-to-end latencies, and therefore are more likely to be among the oldest requests seeking arbitration at a router. Therefore, we propose a router arbitration scheme that uses a message's *distance* as a proxy for its age. The distance can be derived directly from a message's header flit; the header encodes source and destination information, which combined with knowledge of the MN topology, can be translated into a hop count. This information can be precomputed and stored in a very small hardware lookup table for the router arbitration logic. In this case, only about 8 bytes of data are needed so hardware cost is negligible. Using this information, our arbitration performs a weighted round-robin, where the input port selection is weighted by the distance to the memory cube from where the request came from.

### 4.2 The Skip-List Topology

The results from Section 3 showed that the tree topology consistently provides the best overall results. This in of itself is not very surprising as the average hop count increases at a much slower rate than the chain or ring (for increasing MN size). One observation about the tree topology is that the majority of its links, apart from those near the "root," tend to be under-utilized. This is because the total throughput of the MN is ultimately still limited by the single link connecting the MN back to the host memory port.

Since we are evaluating a CPU-centric MN, memory requests and responses are past the coherence ordering point in the system. In directory-based systems, reads to an address with an outstanding write must wait until a write acknowledgment is received at the directory. We can take advantage of this by utilizing "non-coherent" routing. This allows for several interesting network routing policies that are not possible when coherence is required. For example, we are allowed to reorder requests in the network which allows for request priorities to be specified. When injecting to the network, the router output queue could skip over sending write requests at the head of the queue in favor of read requests. Another possibility is to defer non-critical path writes to be routed through lesser used non-optimal routes to free up link utilization for reads. We examine one such topology employing the last technique mentioned.

Taking inspiration from classic data structures, we propose organizing the memory cubes as a "Skip List" [34]. The traditional skip list consists of a regular linked list augmented with additional pointers that provide a path to bypass or short-cut around large sections of the linked list. The "length" of these bypass pointers (i.e., the number of nodes skipped) can be chosen such that the average number of hops to reach a node is logarithmic in the number of nodes in the entire list (same as a tree). Our implementation of the skip-list topology is similar to that of express cubes [14], which proposes additional express channels to reduce network diameter. Fig. 8 shows a 16-node MN organized as a skip list. There is a central sequential chain, analogous to a conventional linked list. The

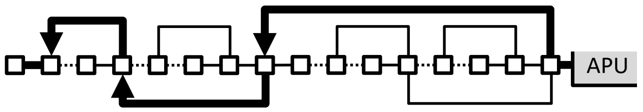

**Figure 8: Example skip-list topology for 16 memory cubes. The bold path shows the route to the farthest memory cube, and the dashed lines indicate links that are only ever used by write requests.**

additional memory cube ports are used to implement the bypass or "skip" links. The set of highlighted (bold) links show how the farthest cube can be reached in only five hops (i.e., logarithmic in the number of cubes).

Using the skip-list topology as is results in a topology that with a lower branching factor than the baseline tree shown in Fig. 3(d), which is a ternary (base-3) tree.[4] Some of the links (shown dashed) would not ever be used because they are not part of the shortest path to any of the cubes. What we instead propose is to differentiate the MN traffic based on read and write requests. For most workloads, read latency directly impacts performance, but writes can typically be handled off of a program's critical path so long as enough aggregate bandwidth is provided. Our skip-list topology routes read requests along the shortest paths to the destination memory cubes, fully utilizing the skip links where possible, different from the routing on express channels. All write requests are routed along the slower, central set of sequential "chain" links. This increases the latency per write request, but removes them from the more performance-sensitive skip links for reads. We effectively trade off some of a tree's fan out (by backing off from a full ternary tree) to shunt off lower-priority write traffic away from the more critical read requests.

As reads and writes to the same address can now take different paths through the MN, there exists the possibility that a read dependent on a write could reach the destination memory cube first, introducing the possibility of a consistency violation (i.e., the read receives a stale value from memory). It is therefore sufficient that the requirement of a directory stalling a read request until a write acknowledgment is received holds true for a skip-list MN.

## 4.3 The MetaCube

The last, most aggressive optimization is to condense the MN into as few cubes as possible. A similar concept to reduce average network latency was adopted by earlier works on bristled [29] and concentrated networks [9]. In our case, we propose leveraging advanced packaging technology to effectively build a "cube of memory cubes," which we call a MetaCube. The MetaCube can use a passive silicon interposer with an interface chip for router or slightly smaller active interposer to integrate multiple memory cubes into the same package. Fig. 9 shows a MetaCube with four DRAM-based memory cubes and an interface chip stacked on a passive silicon interposer.[5]

The external links come from the package pins to the central interface chip in a similar manner to a normal memory cube. The router on the interface chip connects to the different memory stacks through direct point-to-point links across the interposer. Routers

[4]The skip-list ends up with an average branching factor somewhere between 2 and 3.

[5]The same general topology can be implemented on an MCM substrate, with the primary tradeoff being the width of the links between the central interface chips and the individual cubes.

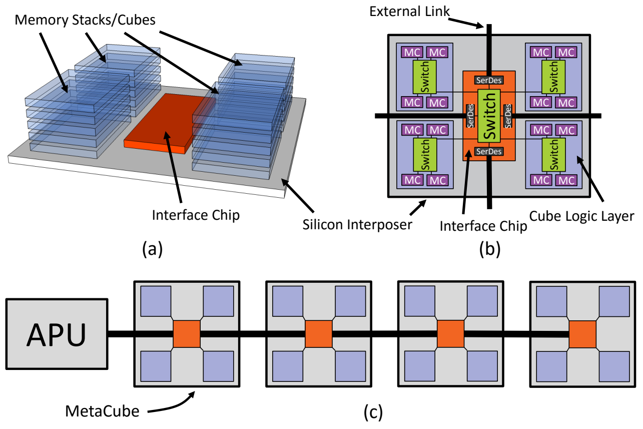

**Figure 9: Example MetaCube organizations. Structural view of a 4-DRAM stack MetaCube, and (b) the block-diagram view of the logic layers.**

on the logic layer of the memory cubes then forward requests to the corresponding memory controllers. Fig. 9(c) shows an example MetaCube MN consisting of four MetaCubes, each internally providing four DRAM memory cubes. This relieves the limitation of 4 ports per memory package by being able to design a high-radix router on an interposer. The maximum number of memory packages in a MetaCube is still limited by the maximum size of an interposer, however, so it is not always possible to create a single MetaCube with an all-to-all network on the interposer. Fig. 9(c) shows an example providing 16 cubes worth of capacity with a greatly reduced worst-case hop count to the furthest memory package.

The MetaCube is not without its own costs. In particular, additional costs must be paid for in terms of the silicon interposer or MCM and interface chips. The package size will also be larger than a conventional memory cube, which will also be more expensive. The power-per-package also increases, which could further increase package-related expenses if more sophisticated cooling solutions are required (although the reduced hop count will at least help reduce interconnect-related power consumption).

## 5 EVALUATION

We explore the design space of MNs using a heterogeneous computing platform consisting of CPU and GPU resources executing a mix of scientific computing workload proxies and high bandwidth GPGPU workloads. We choose these workloads as they provide a high range of memory footprint sizes and bandwidth usage which expose solvable issues with memory networks. Workloads chosen are taken from the AMD SDK [2] and Rodinia [11, 12] suites. Other large footprint workloads such as big data and cloud applications likely exhibit similar issues solvable by the techniques we describe.

The GPU compute units are provisioned similarly to those described for AMD's GCN pipelines [6]. Our system is configured as a 4x8 mesh of GCN pipelines, with a CPU mainly used to perform operating system related system calls such as loading input data and dispatch kernels to the GPU. The key system configuration parameters are listed in Table 2.

Our evaluation implements MNs within the gem5 simulator as part of the Garnet network [3]. We utilize Garnet's default routing

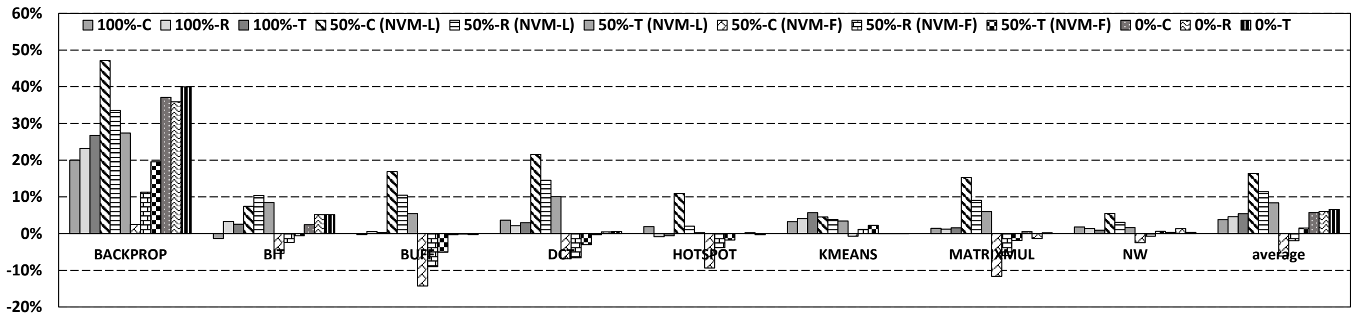

**Figure 10: Speedup comparison using only distance-based arbitration technique. Only baseline topologies are compared to show impact of distance-based arbitration alone.**

**Table 2:** **List of Parameters in Evaluated System.**

| Parameter | Value |
|---|---|
| Compute Units | 32 |
| Clock Speeds | 1GHz (GPU), 3GHz (CPU) |
| GPU L1I$ | 32kB 8-way |
| GPU L1D$ | 16kB 16-way |
| GPU L2$ | 2MB 16-way |
| Memory Ports | 8 |
| Total Memory | 2TB |
| Stack Capacity | 16GB (DRAM), 64GB (NVM) |
| Banks / Stack | 256 |
| DRAM Timings (Major) | tRCD=12ns, tCL=6ns tRP=14ns, tRAS=33ns |
| NVM Timings (Major) | tRCD=40ns, tCL=10ns tWR=320ns, 500MHz |
| DRAM Read/Write NVM Read/Write | 12 pJ/bit [33] 12 pJ/bit / 120 pJ/bit |
| Network Energy | 5 pJ/bit/hop |

implementation to find shortest path routes to each memory package and quadrant of the specific memory controller within the package. This network is modeled as high-speed SerDes links to pipeline packetized data through the MN. For the memory cubes, we evaluate assuming HBM-like memories stacked on a base die with a 4-link router, 4 quadrant design similar to HMC. This allow us to simulate the intra-cube memory timings (i.e., the turnaround time from when a request packet arrives at a cube to when a response packet is sent) using memory timing parameters pulled directly from HBM datasheets.

The internal configuration of each memory cube distributes the memory banks evenly across four quadrants, also similar to HMC. Requests which arrive to a link in the "wrong" quadrant are imposed with a 1ns latency to model intra-cube routing to the correct quadrant and back. Links connecting memory packages are assumed to be narrow 16-bit links running at a frequency of 15GBps [33]. We assume an additional 2ns latency to account for time needed for serialization, scrambling, descrambling, and deserialization circuitry when traversing each SerDes link since these circuits have a non-zero latency even if they are pipelined. We experimented modifying this parameter and found that 2ns made little difference compared to no latency, however larger values (e.g., 10ns) have a large impact on network latency.

Each memory package has a single link connection between another memory package or the host, with four links per memory package. Requests that must route through packages are assumed to be descrambled and deserialized in order to read packet routing information and traverse the internal package switch and therefore incur the additional 2ns latency per hop. Each memory package contains a centralized switch that uses a round-robin arbitration scheme to select between internal quadrants and external links to the MN.

Internally within the system processor, we assume there are 8 ports connected to an external link of a memory cube. Addresses are mapped to a port from within the system processor distributed using address interleaving and hashing techniques for MN load balancing. Requests leave and return through the same port to simplify cache coherence. In order to provide some level of memory parallelism, and in order to stress TB sized memory in simulation within a reasonable turnaround time, addresses are mapped to ports at a 256 byte granularity interleaving. This size was chosen empirically based on a sweep of various mapping sizes. In the presence of spatial locality, larger mapping granularities (e.g., 1024 bytes) caused increases in network latency large enough for performance degradation. The smallest size, 64 bytes, caused reduction in row-buffer hits within the memory cubes.

We estimate the dynamic energy used in the network using average picojoule-per-bit numbers to provide a fairer comparison of NVM and DRAM energies. We assume that network traffic incurs an energy cost at each hop in the network, and we break up NVM values into read and write energies. For network energy, we assume 5pJ/bit for each hop. We assume 12pJ/bit for DRAM [33] and 12pJ/bit for reads 120pJ/bit (10x read energy) for writes with a PCM-based NVM cube. These results do not include static energy, as the static power savings is highly dependent on the underlying process management assumptions (e.g., race-to-idle), although of course NVM provides lower (near-zero) standby power. Note, however, that the memory cube SerDes links will likely still consume non-trivial amounts of power even during "idle" periods because of performance concerns related to the long latencies required to retrain the SerDes links.

## 5.1 Distance-Based Arbitration

We compare our distance-based arbitration approach against the baseline localized round-robin arbitration scheme in each memory cube, with the speedup over round-robin shown in Fig. 10. In order to highlight the impact of the arbitration technique alone, we only compare against the original baseline topologies shown in Fig. 7.

Originally, the study yielded mixed results. In particular, 50% Chain, Ring, and Tree NVM-L topologies show opposite trends compared to 50% Chain, Ring, and Tree NVM-F topologies. One insight from this is that while we attempt to use distance as a proxy

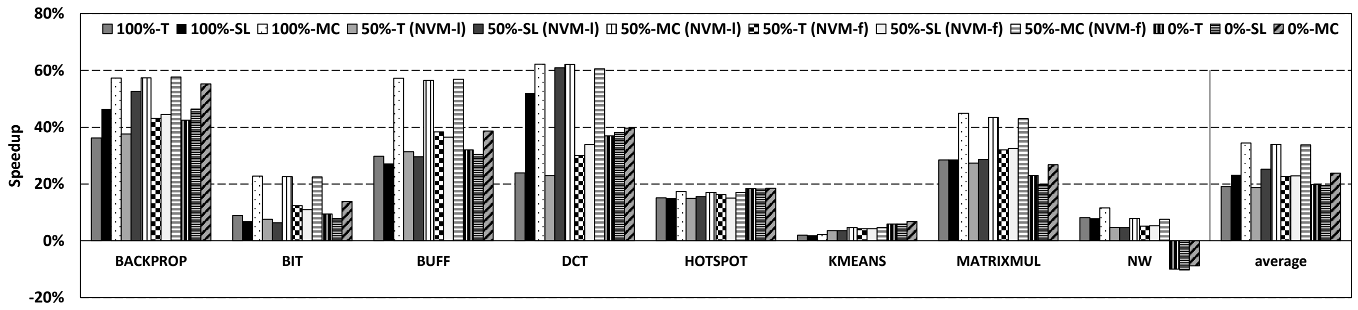

**Figure 11: Comparison of Tree topology compared to our SkipList- and MetaCube-based topologies. The default localized round-robin arbitration is used in these results. All results normalized to 100% Chain.**

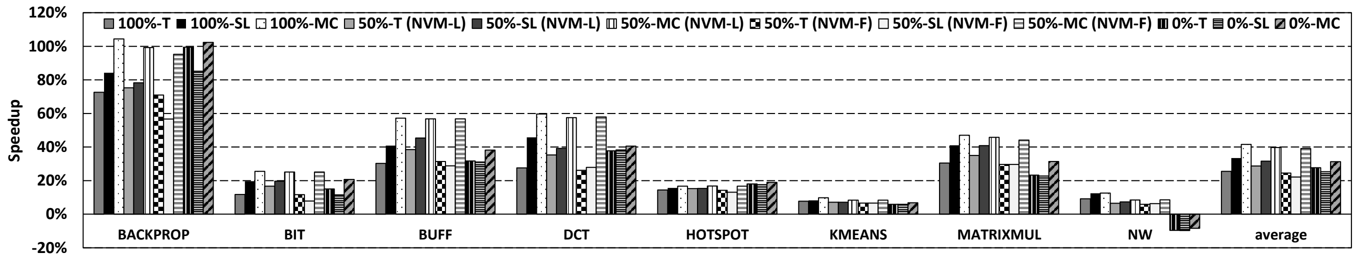

**Figure 12: Overall speedup with all concepts proposed in this paper applied Tree topology, SkipList, and MetaCube using distance-based arbitration. All results normalized to 100% Chain.**

for age, the NVM-F responses will typically be older due to the array latency of NVM. On the other hand, when NVM is placed further from the host processor, the responses returning upstream are nearly guaranteed to be the oldest responses and are much more likely to be selected. In the cases of all-DRAM and all-NVM the average results are very similar. Due to the memory packages having the same latency, the performance results only vary on the distribution of the requests latencies within the cubes (i.e., refresh occurring or unbalanced queuing latency within a specific cube).

In most topologies our distance-based arbitration provides benefits preventing local memory controllers from taking too much weight over request and response packets passing through the memory cube as well as reducing the discrepancies between to- and from-memory latency. Although performance generally increases when using distance-based arbitration, we observed the technique can have adverse effects on topologies such as skip-list and in networks with different mixes of technologies within memory cubes.

In a naive implementation applied to a skip-list, older write requests and responses can be selected over younger read requests and responses due to the longer write path in the network. Similarly, as evidenced by Fig. 10, the technique can have detrimental impact on MNs with NVM placed first, as older responses are predicted to be from the incoming external link, when requests serviced by NVM are actually older.

To combat these issues, we augment our original implementation discussed in Section 4.1 to be aware of both the network topology as well as the memory technology type of the response's source. This allows our distance-based arbitration to better estimate the age of a response. This implementation is applied to our final results shown in Fig. 12.

## 5.2 Skip-List and MetaCube Topologies

Here we compare the tree against the Skip-List and MetaCube topologies by themselves without the usage of distance-based arbitration. The skip-list is effectively a slightly lower arity tree (compared to the default ternary tree), with links to provide extra paths for lower priority requests. The MetaCube is a cluster like topology which similarly aims to reduce hop distance without breaking the constraints of the memory cube package. Results of both topologies are shown in Fig. 11 and Fig. 12

Since the average hop count of skip-list is similar to that of a tree topology, we expect it will perform similarly to that of a tree. On average, the skip-list topology provided only slight benefit over the corresponding tree topology for each DRAM:NVM ratio. For most workloads, contention of read requests due to writes was not great enough to cause the additional write path to make a large impact. The largest average benefit of the skip-list topology is exhibited in the NVM-L topologies. In these configurations, writes which may otherwise fill the queues of local memory vaults are pushed downstream through the network, limiting the amount of performance critical reads blocked in the network at a memory cube's external input port. This phenomenon is not exhibited in most other configurations.

One potential problem with a skip-list topology is always routing writes through the longer path. In some cases, it is beneficial to allow writes to use the skip-paths in presence of very few reads. This allows workloads with either large bursts of write requests or heavy usage of read-modify-writes, for example, to regain some of the performance loss compared to tree topologies. This concept is integrated with our final results in Section 5.3.

Next, a MetaCube topology has the potential to provide an even smaller hop count than either tree or skip-list. This allows for the largest speedup potential of any other given topology across all

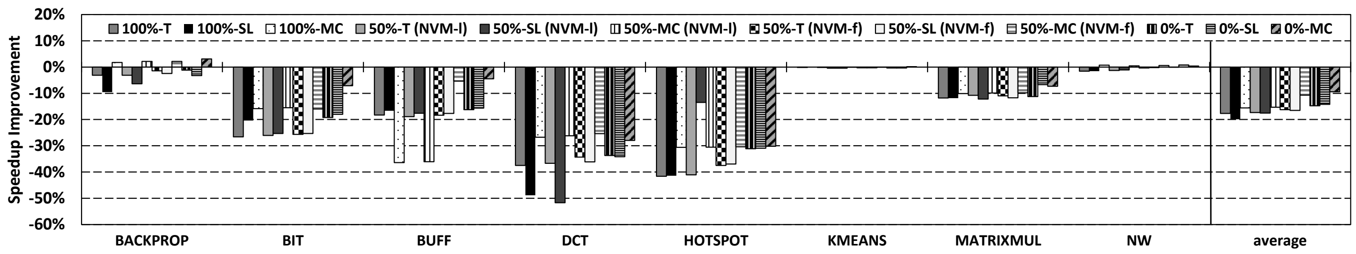

**Figure 13: Average total memory access latency when the host processor has four or eight (baseline) port with a fixed 2TB of memory capacity. Note that the 4-ported system therefore has twice as many packages per port.**

workloads as the cost of additional manufacturing. Since most of the memory round-trip latency consisted of to-memory latency in the majority of workloads, the low hop count of the MetaCube provides significant speedup improvements.

Fig. 11 demonstrates that MetaCubes outperform other types of topology in each simulated run. Different from the previous topologies, the MetaCube is the first topology in which the 100% configurations provided better performance than all topology mixes containing NVM. This occurs since the hop count of a MetaCube configuration is near the threshold where memory array latency, which is higher in NVM, begins to become dominant over both to- and from-memory latency.

However, although Fig. 11 indicates that all-DRAM topologies provide the best performance overall, static power savings from incorporating NVM into the memory system are beneficial in terms of power while yielding nearly the same performance as in the all DRAM case. From our results less than 1% performance degradation was incurred compared to all DRAM MetaCubes.

## 5.3 Combining All Techniques

In this work so far we have evaluated a distance-based arbitration technique, skip-list topology, and MetaCube-based topology. Here we evaluate the combinations of each proposed topology while using distance-based arbitration. Using insights gained from the evaluations throughout this section, we were able to further improve each of the topologies.

In particular, several additions were made to the distance-based arbitration to provide better average performance to topologies. First, our table used to calculate distance was augmented with knowledge of request type priority, allowing writes to be further delayed. This was especially important in the skip-list topology. Next, the same table was augment with knowledge of the memory cube types at each source/destination node allowing it to further weight requests coming from, for example, NVM. Last, we were able to monitor periods of high write traffic at the system port with some hysteresis in order to allow writes to route through shorter network paths in the presence of a skip-list configuration.

The values used to calculate approximate age using distance as a proxy were determined empirically using both average network hop latency and average memory access latency for each cube technology type. Fig. 12 shows the results of this tuning. Compared to the previous results in Fig. 11 we were able to obtain much better average speedup for skip-list in not only 100% MNs, but in 50% NVM-L networks as well (e.g., in BIT and BUFF). This is primarily an artifact of distance-based arbitration's ability to push slower NVM writes further downstream and utilize shorter skip-paths for writes.

As demonstrated in these final results and across many of the previous results, the BACKPROP workload saw the most benefit from each of our experiments. This workload, compared to the others, was by far the most write intensive workload in our suite. In contrast, KMEANS was the most read intensive workload in our suite. However overall each workload in isolation exhibited a respectable increase in performance by employing the methods in this work.

## 6 ANALYSIS

In our evaluations so far, we have seen that MNs are highly sensitive to various configuration parameters, topologies, and workloads. In order to get a better handle of the different parameters which can cause a large impact, we vary the size of the MN in two dimensions. Until now we have simulated a system with 2TB of memory. This is approximately the inflection point where memory cubes begin to overcome the limitations of DDR as discussed in Section 2.1. We reduce the capacity to 1TB in order to decrease the "length" of the MN. Also discussed in Section 2.1 is the limitation on the number of pins available to the host processor package. We study decreasing the number of system ports to decrease the "height" of the MN. Here we examine both of these "dimensions" in order to construct a short sensitivity study.

## 6.1 Number of System Ports

Although eight ports is a reasonable estimate for the available number of pins the host processor could allocate to memory connections[6], some pin-constrained systems may only support fewer ports. Here we investigate the impact of allocating fewer pins to memory by having only four system interfaces to the MNs.

When the number of ports is reduced, the number of memory cube packages per port must be increased in order to provide the same amount of memory capacity. This will generally lead to larger networks, a higher number of average hops, and therefore a performance degradation. Fig. 13 shows the performance loss when decreasing the number of system ports from eight to four. In addition to higher network latency, reducing the number of ports also requires more requests to contend for the same output ports on the processor.

For topologies that grow in a linear fashion (e.g., chain and ring), hop counts double each time the number of ports is reduced by half. This causes the fastest rise in MN latency when compared to other

[6]While current systems only have on the order of four DDR channels per package, recall that HMC-style interfaces are significantly narrower and so eight ports is very reasonable.

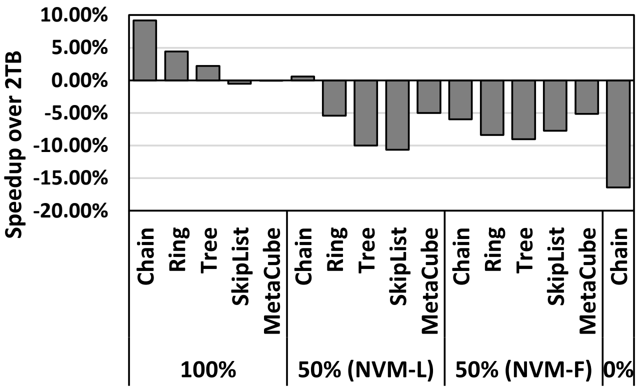

**Figure 14: Average system speedup when moving from 2TB to 1TB.**

topologies. In the case of NVM configurations, 50% NVM-L has the highest degradation. This is again caused by hop count as 50% of requests must be routed to the furthest hop. All-NVM topologies exhibit the lowest degradation on average since these configurations are bound more by memory latency than any other configuration.

Across some workloads, (e.g., BACKPROP and NW) there is a negligible increase (1-3%) in the performance of the MetaCube topologies while the remaining topologies have degraded performance. In these cases the MetaCube provides similar average hop count as the eight-port system with only a marginal increase in latency. The difference in performance for MetaCubes is largely dependent on differences in internal routing within the host processor compared to the eight-port case. Furthermore, workloads such as KMEANS and NW show very little difference in performance.

## 6.2 System Capacity

The total memory capacity of a system has a large impact on network performance and the overall design because the number of memory cubes is dependent upon it. In systems where one specific topology performed the best, smaller-sized systems may have different characteristics. We reduce the capacity of the system by half while keeping the number of memory cubes the same. For the purpose of this study, we assume the workload’s memory footprint is just under the total memory capacity such that it *needs* to access all cubes.

Fig. 14 shows the speedup of a 1TB system over the 2TB baseline. While the reduction in the number of ports intuitively caused the system performance to decrease on average, decreasing the capacity yielded mixed results for each configuration. In the case of 100% topologies, the network latency was reduced while the memory latency remained relatively constant despite each cube servicing twice as many requests. For 50%, the results surprisingly showed a decrease in performance. While the network latency increased by roughly the same amount as the 100% topologies, there was significant increase in the memory’s core latency due to the reduction in memory level parallelism and increased queuing latency in the memory cubes. The largest drop was in the 0% case, as expected, which exhibits performance degradation similar to 50% topologies. This behavior is similar to the conclusion in Section 5.2 where there is an inflection point where memory core latency begins to dominate to- and from-network latency.

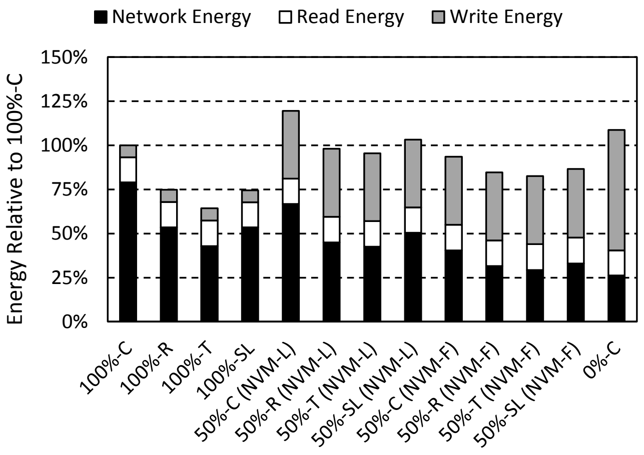

**Figure 15: Breakdown of network (transport) energy and read/write (memory access) energy normalized to a 100%-C MN.**

## 6.3 Energy Analysis

An energy analysis of the MN paints a different picture of the optimal MN. Since NVMs, especially PCM, typically have higher write energy, write-heavy applications consume a large portion of the total energy in the MN. Fig. 15 shows the breakdown of energy usage for network/data movement energy and read/write or memory access energy within the memory package. This breakdown is the average energy utilization of all workloads for each type of MN.

The figure shows there is a significant tradeoff between network energy and memory access energy. For larger networks, the amount of energy required to move data increases linearly with the hop count. This means that DRAM network energy is the largest fraction of total energy in the 100% MNs. On the other end of the spectrum, 0%-C MN reduces network energy by nearly 3x, but the increase in write energy is enough to push the total energy above that of the baseline 100%-C MN. In between these two boundaries, there is more balanced usage in terms of energy.

Topology-wise the tree topology general uses the least amount of energy. Although the skip-list topology emulates a tree-like topology and has a lower average hop count for reads, the additional energy required to route write requests through non-optimal paths increases the network energy by enough to consume more energy than a tree-topology. However, these energy values are highly sensitive to the write energy values used for NVMs. Over time if NVM write energy is reduced, NVM-based MNs may be able to overtake DRAM in both energy usages and performance.

## 7 RELATED WORK

In addition to the work outlined in Section 2.3, Kim et al. also studied the design of MNs in the context of tying together multiple memory cubes with processing-in-memory (PIM) capabilities [25]. As any memory cube/PIM could potentially reference memory locations in any other cube, a full any-to-any MN would be desirable to avoid having to route such requests back through the host processor(s).

Beyond Micron’s seminal Hybrid Memory Cube work [33], others have also advocated for moving future memory systems to adopt more abstract, asynchronous, packet-based interfaces. Resnick and

Ignatowski [36] discuss some of the motivations and possible directions for such future interfaces, particularly packetized abstract interfaces. In our work, we assume a packetized memory system over a more abstract interface compared to that of DRAM, which is in line with the future systems they project. Roberts et al. [37] propose a new memory interface (NMI) that supports point-to-point networks of NMI nodes, nodes with diverse technologies, compatibility for heterogeneous computing (supporting the heterogeneous systems architecture (HSA) specifically), fine-grain logical memory region management, etc. Our proposed architecture can make use of such an interface, as we investigate incorporating different memory technologies, and connect them via point-to-point links.

Cooper-Balis et al. [13] target the problem related to capacity/bandwidth trade-off in traditional DDR-based memory systems by proposing a buffer-on-board design. Although we target a similar problem, our approach is different as we employ MNs to achieve both high bandwidth and high capacity for the system. Zhan et al. [42] demonstrate employing MNs to support in-memory computing and provide latency and power optimizations to their proposed unified MN, targeting an all-to-all connected MN. In contrast, our work does not consider a MN with full connectivity due to reasons explained earlier, and therefore provide non-complementary techniques to tackle the MN latency issues. Azarkhish et al. [8] propose a network for the logic base dies of HMCs. They make use of HMC's base die for employing processing-in-memory (PIM), and their proposed interconnect can provide additional bandwidth to the PIM. While they consider the interconnect design within a cube, we consider the interconnect design for the overall MN. Furthermore, there are multiple efforts that employ near-data computing in the logic base die of memory cubes that are interconnected together [4, 19, 35]. PIM and NDC studies are therefore orthogonal to MNs and both approaches can be complementary. On another front, Akgun et al. [5] propose employing MNs in the silicon interposer of 2.5D integration to enhance the scalability and performance of large capacity designs. Although we propose using silicon interposers for the MetaCube implementation, we only integrate four memory cubes on the interposer and simply use an interface chip to provide direct point-to-point links to achieve low latency. If more memory cubes are to be integrated on the silicon interposer, alternatively, one can decide to implement MNs in lieu of a costly high-radix switch. Rosenfeld [38] analyzes how multiple HMCs perform when they are connected using chain or ring topologies, with various routing algorithms. In our work, we consider other topologies such as skip lists. We also identify performance bottlenecks for such systems, propose arbitration techniques to improve overall MN throughput, and incorporate non-volatile memories to achieve better balance between interconnect and memory array latencies.

## 8 CONCLUSIONS

In this paper, we examined the design of MNs and analyzed sources of performance degradation. Based our analysis, we proposed several mechanisms and MN organizations to address the performance issues. We target globally fair arbitration through distance-based arbitration, which estimates the age of a request and provide several insights behind implementation of such a scheme. We introduced a non-coherent skip-list-based topology able to provide performance matching or exceeding, on average, that of the lowest hop count tree topology. To reduce MN size/diameter, we consider two approaches: the first uses the higher density of NVM and the second leverages advanced packaging technologies to create very high-capacity MetaCubes, both of which increase per-package capacity and reduce MN size. These provide effective techniques to improve the performance of future MNs to support modern servers, supercomputers, and high-end workstations with terabyte-scale memory capacities.

While this paper focused on MNs for a particular family of system organizations (specifically where each of the host processor's memory ports covers a disjoint portion of the physical address space), the insights learned and techniques proposed should extend to other types of MNs. For example, distributed arbitration to increase fairness/throughput and read-write differentiated routing could both be promising for multi-GPU systems [26] or networks of PIMs [25], optimizations and future non-coherent topologies, and the ability to cluster memory network-like networks into MetaCube like devices.

## ACKNOWLEDGMENTS

AMD, the AMD Arrow logo, Radeon, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

Akgun and Xie were supported in part by DOE grant DE-SC0013553 and NSF grants 1500848/1533933.

## REFERENCES

[1] The Machine: A new kind of computer. https://www.labs.hpe.com/the-machine.

[2] Inc. Advanced Micro Devices. AMD SDK. http://developer.amd.com/tools-and-sdks.

[3] Niket Agarwal, Tushar Krishna, Li-Shiuan Peh, and Niraj K Jha. 2009. GARNET: A detailed on-chip network model inside a full-system simulator. In *Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on*. IEEE, 33–42.

[4] Junwhan Ahn, Sungpack Hong, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. 2015. A scalable processing-in-memory accelerator for parallel graph processing. In *2015 Intl. Symp. on Computer Architecture*. ACM, 105–117.

[5] Itir Akgun, Jia Zhan, Yuangang Wang, and Yuan Xie. 2016. Scalable Memory Fabric for Silicon Interposer-Based Multi-Core Systems. In *2016 Intl. Conf. on Computer Design*.

[6] AMD Advanced Micro Devices, Inc. 2012. AMD Graphics Core Next (GCN) Architecture. (June 2012). https://www.amd.com/Documents/GCN_Architecture_whitepaper.pdf.

[7] Arstechnica. HBM3: Cheaper, up to 64GB on-package, and terabytes-per-second bandwidth. http://arstechnica.com/gadgets/2016/08/hbm3-details-price-bandwidth/.

[8] Erfan Azarkhish, Davide Rossi, Igor Loi, and Luca Benini. 2014. A Logic-base Interconnect for Supporting Near Memory Computation in the Hybrid Memory Cube. In *2nd Workshop on Near-Data Processing*. Cambridge, UK.

[9] James Balfour and William J. Dally. 2006. Design Tradeoffs for Tiled CMP On-chip Networks. In *2006 Intl. Conf. on Supercomputing (ICS '06)*. ACM, New York, NY, USA, 187–198. https://doi.org/10.1145/1183401.1183430

[10] Paul Benson. 2009. Dell$^{TM}$ PowerEdge$^{TM}$ Servers 2009 - Memory. *Dell Technical Whitepaper* (February 2009).

[11] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron. 2009. Rodinia: A Benchmark Suite for Heterogeneous Computing. In *IEEE Intl. Symp. on Workload Characterization*.

[12] S. Che, J. W. Sheaffer, M. Boyer, L. G. Szafaryn, L. Wang, and K. Skadron. 2010. A Characterization of the Rodinia Benchmark Suite with Comparison to Contemporary CMP Workloads. In *IEEE Intl. Symp. on Workload Characterization*.

[13] Elliott Cooper-Balis, Paul Rosenfeld, and Bruce Jacob. 2012. Buffer-on-board Memory Systems. In *39th Intl. Symp. on Computer Architecture*. Portland, OR, 392–403.

[14] W. J. Dally. 1991. Express cubes: improving the performance of k-ary n -cube interconnection networks. *IEEE Trans. Comput.* 40, 9 (Sep 1991), 1016–1023. https://doi.org/10.1109/12.83652

[15] Dell Corporation. 2016. *Memory Speeds and Population*. Technical Report. http://www.dell.com/learn/us/en/2684/campaigns/ memory-speeds-and-population.

[16] Xiangyu Dong, Yuan Xie, Naveen Muralimanohar, and Norman P. Jouppi. 2010. Simple but Effective Heterogeneous Main Memory with On-Chip Memory Controller Support. In *SC*.

[17] Xiangyu Dong, Yuan Xie, Naveen Muralimanohar, and Norman P. Jouppi. 2010. Simple but Effective Heterogeneous Main Memory with On-Chip Memory Controller Support. In *2010 ACM/IEEE Int'l Conf. for High Performance Computing, Networking, Storage and Analysis (SC '10)*. IEEE Computer Society, Washington, DC, USA, 1–11. https://doi.org/10.1109/SC.2010.50

[18] Brinda Ganesh, Aamer Jaleel, David Wang, and Bruce Jacob. 2007. Fully-Buffered DIMM Memory Architectures: Understanding Mechanisms, Overheads and Scaling. In *Proceedings of the 2007 IEEE 13th International Symposium on High Performance Computer Architecture (HPCA '07)*. IEEE Computer Society, Washington, DC, USA, 109–120. https://doi.org/10.1109/HPCA.2007.346190

[19] M. Gao, G. Ayers, and C. Kozyrakis. 2015. Practical Near-Data Processing for In-Memory Analytics Frameworks. In *2015 Intl. Conf. on Parallel Architectures and Compilation Techniques*. 113–124.

[20] HMCC. Hybrid Memory Cube Specification 1.0. http://hybridmemorycube.org/specification-download/.

[21] HMCC. 2014. Hybrid Memory Cube Specification 2.1. http://www.hybridmemorycube.org/specification-v2-download-form/. Online.

[22] Supreet Jeloka, Reetuparna Das, Ronald G. Dreslinski, Trevor Mudge, and David Blaauw. 2014. Hi-Rise: A High-Radix Switch for 3D Integration with Single-Cycle Arbitration. In *47th Intl. Symp. on Microarchitecture*. Cambridge, UK, 471–483.

[23] Xiaowei Jiang, Niti Madan, Li Zhao, M. Upton, R. Iyer, S. Makineni, D. Newell, Y. Solihin, and R. Balasubramonian. 2010. CHOP: Adaptive Filter-Based DRAM Caching for CMP Server Platforms. In *HPCA-16*.

[24] G. Kim, J. Kim, J-H. Ahn, and J. Kim. 2013. Memory-centric system interconnect design with hybrid memory cubes. In *Intl. Conf. on Parallel Architectures and Compilation Techniques*.

[25] Gwangsun Kim, John Kim, Jung Ho Ahn, and Yongkee Kwon. 2014. Memory Network: Enabling Technology for Scalable Near-Data Computing. In *2nd Workshop on Near-Data Processing*. Cambridge, UK.

[26] Gwangsun Kim, Minseok Lee, Jiyun Jeong, and John Kim. 2014. Multi-GPU System Design with Memory Networks. In *47th Intl. Symp. on Microarchitecture*. Cambridge, UK, 484–495.

[27] Emre Kultursay, Mahmut Kandemir, Anand Sivasubramaniam, and Onur Mutlu. 2013. Evaluating STT-RAM as an Energy-efficient Main Memory Alternative. In *2013 Intl. Symp. on Performance Analysis of Systems and Software*. IEEE, 256–267.

[28] Benjamin C. Lee, Engin Ipek, Onur Mutlu, and Doug Burger. 2009. Architecting Phase Change Memory As a Scalable Dram Alternative. In *2009 Intl. Symp. on Computer Architecture (ISCA '09)*. ACM, New York, NY, USA, 2–13. https://doi.org/10.1145/1555754.1555758

[29] José F. Martínez, Josep Torrellas, and José Duato. 1999. Improving the Performance of Bristled CC-NUMA Systems Using Virtual Channels and Adaptivity. In *1999 Intl. Conf. on Supercomputing (ICS '99)*. ACM, New York, NY, USA, 202–209. https://doi.org/10.1145/305138.305194

[30] Mitesh Meswani, Sergey Balgodurov, David Roberts, John Slice, Mike Ignatowski, and Gabriel Loh. 2015. Heterogeneous Memory Architectures: A HW/SW Approach for Mixing Die-stacked and Off-package Memories. In *21st Intl. Symp. on High Performance Computer Architecture*. San Francisco, CA, 126–136.

[31] Justin Meza, Jichuan Chang, HanBin Yoon, Onur Mutlu, and Parthasarathy Ranganathan. 2012. Enabling Efficient and Scalable Hybrid Memories Using Fine-granularity DRAM Cache Management. *Computer Architecture Letters* 11, 2 (2012), 61–64.

[32] R. Nair, S.F. Antao, C. Bertolli, P. Bose, J.R. Brunheroto, T. Chen, C. Cher, C.H.A. Costa, J. Doi, C. Evangelinos, B.M. Fleischer, T.W. Fox, D.S. Gallo, L. Grinberg, J.A. Gunnels, A.C. Jacob, P. Jacob, H.M. Jacobson, T. Karkhanis, C. Kim, J.H. Moreno, J.K. O'Brien, M. Ohmacht, Y. Park, D.A. Prener, B.S. Rosenburg, K.D. Ryu, O. Sallenave, M.J. Serrano, P.D.M. Siegl, K. Sugavanam, and Z. Sura. 2015. Active Memory Cube: A processing-in-memory architecture for exascale systems. *IBM Journal of Research and Development* 59, 2/3 (2015), 17–1.

[33] J. Thomas Pawlowski. 2011. Hybrid Memory Cube: Breakthrough DRAM Performance with a Fundamentally Re-Architected DRAM Subsystem. In *Hot Chips 23*.

[34] W. Pugh. 1990. Skip lists: A probabilistic alternative to balanced trees. *Communications of the Association for Computing Machinery* 33, 6 (June 1990), 668–0676.

[35] S.H. Pugsley, J. Jestes, Huihui Zhang, R. Balasubramonian, V. Srinivasan, A. Buyuktosunoglu, A. Davis, and Feifei Li. 2014. NDC: Analyzing the impact of 3D-stacked memory+logic devices on MapReduce workloads. In *2014 Intl. Symp. on Performance Analysis of Systems and Software*. IEEE, 190–200.

[36] David R. Resnick and Mike Ignatowski. 2014. *Proposing an Abstracted Interface and Protocol for Computer Systems*. SAND2014 15795. Sandia National Laboratories.

[37] David Roberts, Amin Farmahini-Farahani, Kevin Cheng, Nathan Hu, David Mayhew, and Michael Ignatowski. 2015. NMI: A New Memory Interface to Enable Innovation. In *Hot Chips*. Aspen, Colorado. http://www.hotchips.org/wp-content/uploads/hc_archives/hc27/HC27.26-Posters/HC27.26.p40-NMI-Memory_Interface-Roberts-AMD.pdf

[38] Paul Rosenfeld. 2014. *Performance Exploration of the Hybrid Memory Cube*. Ph.D. Dissertation. University of Maryland, College Park.

[39] Samsung Electronics Corp. 2016. Samsung Begins Mass Producing World's Fastest DRAM - Based on Newest High Bandwidth Memory (HBM) Interface. (Jan. 19, 2016). http://news.samsung.com.

[40] W. Song, H. J. Jung, J. Ahn, J. Lee, and J. Kim. 2016. Evaluation of Performance Unfairness in NUMA System Architecture. *IEEE Computer Architecture Letters* PP, 99 (2016), 1–1. https://doi.org/10.1109/LCA.2016.2602876

[41] ChunYi Sun, Edgar A. Leon, Gabriel H. Loh, David Roberts, Kirk Cameron, Dimitrios S. Nikolopoulos, and Bronis S. de Supinkski. 2015. HpMC: An Energy-aware Management System of Multi-level Memory Architectures. In *2015 Intl. Symp. on Memory Systems*. Washington DC.

[42] Jia Zhan, Itir Akgun, Jishen Zhao, Al Davis, Paolo Faraboschi, Yuangang Wang, and Yuan Xie. 2016. A Unified Memory Network Architecture for In-Memory Computing in Commodity Servers. In *2016 Intl. Symp. on Microarchitecture*. IEEE.