Understanding and Optimizing Power Consumption in Memory Networks

Xun Jian
*University of Illinois
at Urbana-Champaign
Champaign, USA
xunjian1@illinois.edu*

Pavan Kumar Hanumolu
*University of Illinois
at Urbana-Champaign
Champaign, USA
hanumolu@illinois.edu*

Rakesh Kumar
*University of Illinois
at Urbana-Champaign
Champaign, USA
rakeshk@illinois.edu*

***Abstract*—**
As the amount of digital data the world generates explodes, data centers and HPC systems that process this big data will require high bandwidth and high capacity main memory. Unfortunately, conventional memory technologies either provide high memory capacity (e.g., DDRx memory) or high bandwidth (GDDRx memory), but not both. Memory networks, which provide both high bandwidth and high capacity memory by connecting memory modules together via a network of point-to-point links, are promising future memory candidates for data centers and HPCs. In this paper, we perform the first exploration to understand the power characteristics of memory networks. We find idle I/O links to be the biggest power contributor in memory networks. Subsequently, we study idle I/O power in more detail. We evaluate well-known circuit-level I/O power control mechanisms such as rapid on off, variable link width, and DVFS. We also adapt prior works on memory power management to memory networks. The adapted schemes together reduce I/O power by 32% and 21%, on average, for big and small networks, respectively. We also explore novel power management schemes specifically targeting memory networks, which yield another 29% and 17% average I/O power reduction for big and small networks, respectively.

***Keywords*-Memory Power Management, Hybrid Memory Cube, Point-to-point Network, High-speed Memory I/O**

I. Introduction

Processor architectures for data centers and HPC systems have become increasingly throughput-oriented and many core. As the number of cores per processor increases, so does the amount of memory capacity and bandwidth required per processor to match the core count. Conventionally, memory systems scale in capacity by adding more memory modules to shared memory buses. However, multiple devices sharing a bus negatively impacts I/O signal integrity [1] and thus limits memory capacity and bandwidth [2].

Instead of sharing multiple memory modules on the same memory bus, a memory network [3], [4] connects one memory module to another via a point-to-point (P2P) link dedicated to each pair of modules. By adding memory modules via an expandable network of P2P links, a memory network provides similar or higher capacity scaling as conventional high-capacity DDRx memories. However, by only connecting a minimum number of (i.e., two) devices per link, P2P links improves signal integrity [1] and, therefore, allow high I/O frequencies and thus bandwidth. For example, the Hybrid Memory Cube (HMC) [5], an emerging memory network technology, increases I/O frequency by up to 8X compared to DDR4, the latest generation memory with a shared bus interface. Due to providing high bandwidth and capacity, memory networks will likely benefit future data centers and HPC systems.

In addition to high memory capacity and bandwidth, large-scale systems also require high memory energy efficiency. Memory systems consume 25% - 40% of the total data center power [6]. In a projection based on the current level of power efficiency, memory power alone will consume 3.5X the total power budget of future exascale systems [7].

In this paper, we perform the first exploration to understand the power characteristics of memory networks. We study different memory network topologies and sizes and report a detailed breakdown of idle and active power consumption among different memory components. We find that idle I/O power is the biggest source of memory network power consumption. Subsequently, we study idle I/O power in more detail. We evaluate various well-known I/O power control mechanisms such as DVFS, variable link width (VWL), rapid on off (ROO), and their combinations. We adapt prior works on memory power management to manage I/O power in memory networks. Since no prior power management works have been proposed in the context of memory networks, we refer to our adaptation of prior schemes collectively as *network-unaware management*. We find that network-unaware management reduces I/O power by 32% and 21%, on average, for big and small networks, respectively. However, idle I/O power still remains the top power contributor. Consequently, we explore and propose novel network-aware management policies; they provide 29% and 17% I/O power reduction for big and small networks, respectively, compared to network-unaware management. This paper makes the following contributions:

- The first exploration to understand memory network power characteristics. We identify idle I/O as the biggest power contributor.
- The evaluation of various circuit-level I/O power control mechanisms, such rapid on off, variable link width, DVFS, and combinations thereof, and analysis of their

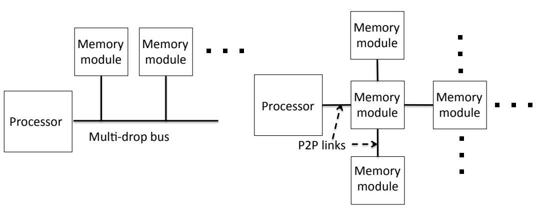

Figure 1. Left: conventional memory systems. Right: Memory networks.

relative effectiveness.

- The first adaptation of prior works on memory power management to memory networks; average I/O power is reduced by 32% and 21% for big and small networks, respectively. We also show that the choice of network topology (e.g., linear, tree, etc.) significantly affects the effectiveness of power management.
- The first network-aware memory power management, which yields another 29% and 17% average I/O power reduction for big and small networks, respectively.

II. Background

Memory system capacity is increased by adding memory modules. Different types of memory systems differ in terms of how memory modules are added to the system.

A. Conventional Memory Systems

Conventional memory systems increase memory capacity by adding more memory modules to shared memory buses, as shown in the left half of Figure 1. However, increasing the number of electrical devices on a bus reduces signal integrity [1], [2]; this limits the number of memory modules that can be reliably connected to a bus and, therefore, the maximum memory capacity per bus. Having more memory buses per processor increases the maximum capacity and bandwidth of a memory system, but requires more I/O pins on the processor chip. Unfortunately, the number of I/O pins per processor is limited [8] since I/O pins increase processor area and, therefore, cost, which in turn limits the maximum capacity and bandwidth of conventional memory systems. As such, conventional memory systems either provide high bandwidth but low capacity (e.g., GDDRx memory with P2P data I/O pins that allow up to 14 Gbps but not sharing) or high capacity but low bandwidth (e.g., DDRx memory with data I/O pins that allow up to only 3.2 Gbps but sharing).

B. Memory Network

In addition to memory, each memory module in a memory network also contains buffering/routing logic to communicate with other memory modules via high-speed P2P links. This allows a memory network to increases its capacity via a network of P2P links, as shown in the right half of Figure 1. This expandable network of high-speed P2P links provides both high bandwidth and high capacity memory.

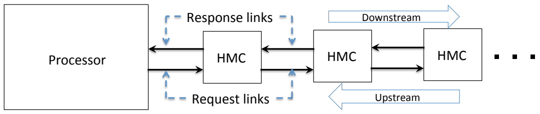

Figure 2. HMC network example.

A well-known example of a memory network module is the HMC[1]. An HMC consists of multiple DRAM dies stacked on top of a logic die using TSVs (through-silicon-vias) [5]. The DRAM dies provide memory capacity, while the logic die implements network routing logic and I/O circuitry [5]. An HMC can provide up to 30 Gbps [5] of I/O frequency; in comparison, current DDR4 DIMMs, the latest memory modules found in conventional memory systems, only support an I/O frequency of up to 3.2Gbps per lane [10]. HMCs also improve memory energy per bit by 3X compared to DDR4 [11]. Due to the many benefits of HMCs, we explore memory networks in the context of HMCs.

Figure 2 shows an example of an HMC network. A network of HMCs communicates via unidirectional links and a packet-based protocol [5], which are commonly used to support high-speed I/O communication; they are also used by buffer-on-board memory systems [9], for example. We refer to unidirectional links that send data away from and toward the processor as *request links* and *response links*, respectively. A read request packet consists of a single 16B *flit* (i.e., minimum traffic flow unit), while write request and read response packets contain five flits, assuming 64B lines.

III. Analyzing Power Characteristics of Memory Networks

In this paper, we perform the first exploration to understand the power characteristics of memory networks.

A. Network Topologies

We evaluate minimally connected network topologies. For a given set of networked memory modules, a minimally connected topology minimizes the average and worst-case hop distances between the processor and its memory modules by connecting every available link to a new module, instead of spending them on already connected modules. A minimally connected topology is also acyclic and, therefore, does not require deadlock or livelock avoidance logic.

The HMC standard supports high-radix HMCs with four full links (i.e., eight unidirectional links) and low-radix HMCs with two full links [5]. We evaluate networks consisting of only high-radix HMCs, of only low-radix HMCs, and a mixture of both. Figure 3 shows the topologies we examine. We evaluate the ternary tree, which minimizes

[1] As another example, a memory network module could be a few high bandwidth (e.g., GDDRx) memory chips attached to a buffer/router chip, such as a buffer-on-board chip [9], which connects to other such memory network modules via P2P links.

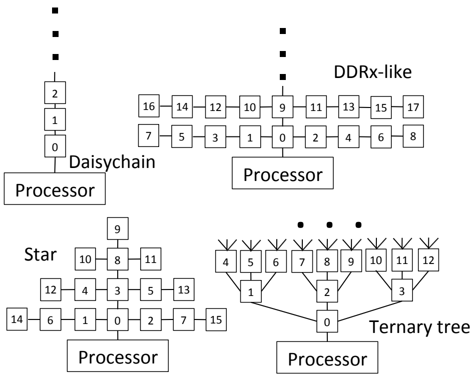

Figure 3. Topologies studied.

network hop distance, using high-radix HMCs. We evaluate the daisy chain using only low-radix HMCs to minimize HMC area. For a mixture of both types of HMCs, we evaluate the star topology which grows by adding rings of nodes equidistant from the processor; for smaller network sizes, star offers the same hop distances as the ternary tree while requiring fewer high-radix HMCs. We also evaluate another mixed-HMC topology that scales in capacity by adding rows of memory packages, similar to how DDRx DIMMs scale in capacity by adding rows or ranks of DDRx memory packages, for potential ease of adoption; we refer to this as the *DDRx-like* topology.

B. HMC Modeling

We use the HMC power model in [12] to evaluate high-radix HMCs. In [12], high radix HMCs with 12.5Gbps I/O data rate per lane consume 13.4W of peak power. [12] models peak power by attributing 43%, 22%, and 35% of the peak power of an HMC to the peak power of the DRAM dies, the logic part of the logic die (which we simply refer to as *logic*), and the I/O links, respectively; for idle power, the DRAM dies consume 10% of its peak power when idle, logic consumes 25% of its peak power when idle, while idle I/O power (i.e., when not transmitting application data) is same as active I/O power. Idle and active I/O power are similar because high-speed links need to continuously transmit data even when idle to maintain synchronization between the transmitter and receiver [5], [18]. To evaluate low-radix HMCs, we assume that memory peak power is proportional to memory bandwidth and, therefore, assume the peak power of low-radix HMCs to be half of 13.4W; we also assume the same relative power breakdown as above.

We use DRAMSim2 and the parameters in Table I to model the performance of DRAM array accesses and modify GEM5 to model I/O link performance. For the I/O links, we assume that each link controller contains 128 buffer entries and prioritize reads over writes, as writes do not typically lie along the critical path of execution. We model 3.2ns SERDES link latency. To model routing within an HMC, we assume a pipelined router with 0.64ns (the minimum transfer latency of a single flit over the evaluated links) clock period and four cycles latency.

Table I
HMC DRAM ARRAY PARAMETERS

Capacity per HMC/vaults per HMC	4GB/32
Vault data rate/IO width/buffer entries	2Gbps/X32/16
page policy/line address mapping	close/interleaved
tCL/tRCD/tRAS/tRP/tRRD/tWR(ns)	11/11/22/11/5/12

Table II
PROCESSOR MICROARCHITECTURE

Core	16 cores, 3GHz, 2-issue OOO 64 ROB entries, 64B cache line size
L1 d-cache, i-cache	2-way, 64kB, 1 cycle
Private L2 cache	8-way, 512kB, 3 cycles
Shared L3 cache	32-way, 32MB, 20 cycles

C. Processor and Workloads

We model a 16-core X86 processor in GEM5. Detailed architectural parameters used for GEM5 simulation are listed in Table II. Since different memory channels are physically independent from one another and bandwidth utilization is often uniformly distributed across channels by interleaving adjacent memory across channels [13], we evaluate a single HMC channel with little loss of generality; we leave the exploration of power implication of any potential inter-channel interactions to future work.

We evaluate seven HPC workloads and seven cloud computing workloads using full-system simulation. The average memory footprint of all of our workloads is 17GB. The HPC workloads include 16-threaded $ua.D$, $lu.D$, $bt.D$, $sp.D$, $cg.D$, $mg.D$, and $is.D$ from NASBench. The cloud workloads are mixed application workloads each consisting of four applications, at least one of which is a parallel SPLASH2X benchmark and the remaining of which are SPEC2006 benchmarks; only native or reference inputs are used. Each parallel application runs on four threads, while each SPEC application runs as four independent instances. Table III details the workload composition for the mixed workloads; the applications in each workload appear in the order of their invocation, which determines memory allocation as we delay the invocation of each subsequent application or instance by one simulated second. We fast forward each workload until all multi-threaded application(s) in each workload have completed their initialization, as

Table III
MIXED WORKLOAD COMPOSITION

mixA	4 bwaves, 4 cactusADM, 4 wrf, ocean_cp
mixB	4 mcf, 4 GemsFDTD, 4T barnes, 4T radiosity
mixC	4 omnetpp, 4 mcf,4 wrf, 4T ocean_cp
mixD	4 sjeng, 4 cactusADM,4T radiosity, 4T fft
mixE	4 cactusADM, 4 sjeng,4 wrf, 4T fft
mixF	4 cactusADM, 4 bwaves,4 sjeng, 4T fft
mixG	4 mcf, 4 omnetpp,4 astar, 4T fft

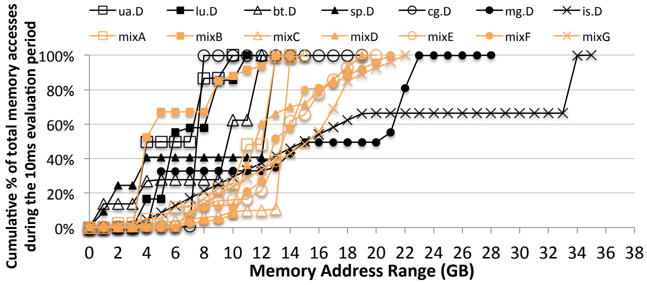

Figure 4. Workload memory access characteristics.

indicated by the simulated OS console output, and then by another 20 simulated seconds to warm up the caches; the total fast-forward period is 73 seconds, on average, and up to 218 seconds. We evaluate each workload in cycle-accurate mode for the next 10ms of simulated time. Since there is a wide range in the memory footprint of our workloads, each workload is evaluated for a memory network whose size matches the memory footprint of the workload. For our evaluations, we map the i^{th} contiguous 4GB of physical pages (we use 4GB HMCs in our evaluation) to the i^{th} HMC in the network (see Figure 3 for the location of HMC i); therefore, the average number of HMCs per workload is $\lceil 17/4 \rceil = 5$. Since memory networks can support a large number of memory modules, we also perform a big network study by mapping the i^{th} contiguous 1GB to the i^{th} HMC. Figure 4 shows the cumulative fraction of memory accesses by the i^{th} gigabyte of memory address space of each workload during the cycle-accurate simulation interval; memory traffic distribution within the networks can be deduced through Figure 3 and 4.

D. Key Findings

Figure 5 shows the average power consumption per memory module; “Idle I/O” and “Active I/O” are calculated as total energy consumed by links while idle and active, respectively, divided by the total number of HMCs and by time. The first key observation is that I/O power is the highest power contributor in memory networks; I/O consumes, on average, 73% of the memory network power. There are two reasons why I/O power dominates. First, each memory request in a memory network accesses a DRAM die once but traverses multiple links (see Figure 6), which contributes to I/O having much higher power than DRAM. Second, even within a single module, a major fraction of energy per memory access is due to I/O. For example, moving data off package consumes roughly twice as much energy per bit as moving data within package from DRAMs to the logic die [14]. As another example, I/O consumes 40% of access energy in DDR3 DRAMs [15].

There are several reasons why I/O power is high even for a single memory module. First, due to the high area cost of I/O pins, memory I/O width is often many times narrower than the DRAM array data bus width. Therefore, I/O must operate at much higher frequencies than the DRAM arrays to match the bandwidth and thus consume high power. Another reason is that significant power is required to maintain signal integrity for off-chip communication. For example, transmitter output impedance must closely match the characteristic impedance of the off-chip transmission channel (i.e., a PCB trace) to minimize I/O signal reflection; impedance matching is typically implemented by terminating the transmitter output using a similar impedance (see Figure 7). Unfortunately, PCB traces typically have low characteristic impedance (around 50Ω) to keep ohmic power loss low; the matching low impedance termination results in a low impedance connection to ground and thus high power consumption (e.g., $(1V)^2/(50\Omega) = 20mW$ per lane assuming 1V signal voltage or $20 * 32 = 0.64W$ per HMC full link). Receivers also require similar overheads.

The second key observation is that when further breaking down power into idle and active, idle I/O power is the highest power contributor in memory networks; it accounts for 53% and 67% of total memory network power in the small and big network studies, respectively (see black and orange points in Figure 8). To understand the sources of high idle I/O power, we define channel utilization as the bandwidth utilization of the full link that connects the processor to a memory network and define link utilization as the average bandwidth utilization across all links in a network. Figure 9 shows the channel and link utilizations of the different workloads under different topologies. As expected, the fraction of total network power taken up by

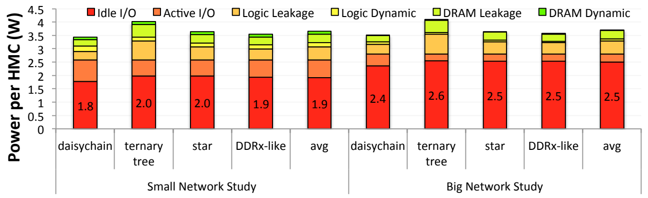

Figure 5. Average power breakdown of an HMC in a network.

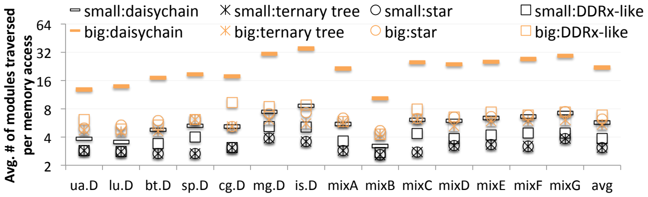

Figure 6. Links traversed per memory access.

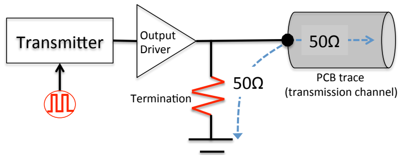

Figure 7. Some major causes of high I/O power.

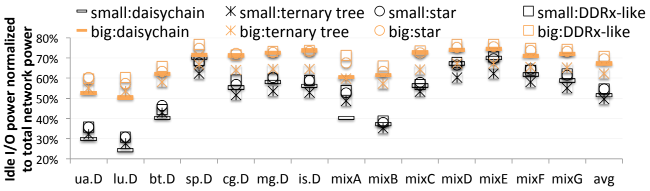

Figure 8. Idle I/O power consumption by workload.

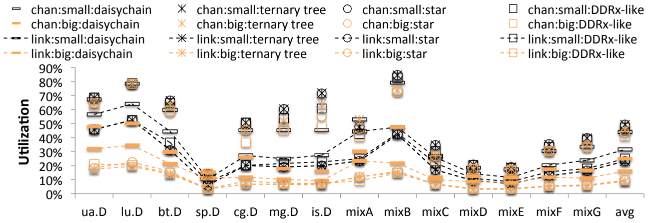

Figure 9. Average channel and link utilization.

idle I/O power increases when channel bandwidth utilization decreases; for example, under $sp.d$, which has the lowest channel utilization, idle I/O power is also the highest (see Figure 8). However, idle I/O power still accounts for 50% of total memory network power even for $mixB$, which has 75% average channel utilization; in fact, the average channel utilization across our evaluated workloads is high - 43%. Idle I/O power remains high despite high channel utilization because memory traffic attenuates across the network; as such, average link utilization continues to be low (see the dotted lines in Figure 9) even when channel utilization is high (see data points without the dotted lines).

Since idle I/O power accounts for over half of total memory network power, we will explore idle I/O power management for memory networks in the rest of the paper.

IV. I/O Power Control Mechanisms

Many circuit-level mechanisms exist to dynamically reduce I/O power during low utilization, with different power and performance tradeoff characteristics. Below we discuss the mechanisms we study in this paper.

A. Rapid On Off Links

In conventional DRAMs, I/O is commonly turned off (i.e., put in an inaccessible low power state) when idle to reduce idle power. However, once turned off, an I/O link needs to first be woken up before it can be accessed again, leading to performance overheads. Conventional DRAMs typically require 10-25ns to wake up the I/O [10], depending on the off-state power. Both the wakeup latency and off state power should ideally be zero. This is difficult for two reasons, however. First, to wake up a link, the link transmitter must first resynchronize with the receiver before data can be reliably transmitted. Second, increasing the current from the off state level (where current $i \approx 0$) to nominal operating level typically requires many nanoseconds since current change is opposed by a back electromagnetic force (EMF) that is proportional to the rate of current change (i.e., $EMF = Ldi/dt$); since I/O operating current is high (due to the high operating power of I/O), di/dt is also high.

To model ROO, we assume 14ns wakeup latency [16] per unidirectional[2] link and 1% power when off [16]. We also examine 20ns wakeup latency [18] for sensitivity analysis.

B. Dynamic Voltage Frequency Scaling

Another mechanism to reduce I/O power during low utilization is to keep the link on but reduce I/O bandwidth via dynamic voltage frequency scaling (DVFS) [16]. Since dynamic energy is $\propto CV^2f$, DVFS reduces dynamic energy per bit in addition to idle power, unlike ROO, which only reduces idle power. DVFS avoids the long wakeup latency of ROO; however, DVFS increases SERDES latency since SERDES - the circuit that converts parallel data input into a high-speed serial data stream for transmission - is clocked by the I/O clock as well [16]. DVFS also requires significant latency to adjust voltage since changing voltage requires a current draw that is proportional to the rate of change (i.e., i=Cdv/dt); to adjust voltage quickly (e.g., small dt), a more heavy-weight on-chip voltage regulator capable of supplying much higher current than is needed during regular operations is required, which incurs area and power overheads. As such, on-chip voltage regulators typically require long latency to adjust voltage (e.g., 0.5us [19]). Due to the long voltage scaling latency, DVFS can incur higher queuing latency overheads than ROO for a long burst of accesses.

We model link voltage switching latency as 0.5us. To ensure connectivity during voltage scaling, we assume that the sixteen lanes per link are split into two bundles of eight lanes each, where each bundle has separate voltage rails such that DVFS is only applied to one bundle at a time. This leads to up 3us to complete voltage scaling for a link; 1 us to reduce operating link width to half (see Section IV-C), 1 us to DVFS the two bundles, and 1 us to resume full link width. We model the performance and power of DVFS during steady-state operations using [16] by evaluating DVFS modes that provide 100%, 80%, 50%, and 14% bandwidth and provide 0%, 30%, 65%, and 92% power reduction, respectively; the modes are selected such that each subsequent mode provides roughly equal amount of total link power reduction (e.g., 30%) as the previous mode. The lowest power mode operates only a single bundling of eight lanes at the minimum I/O operating voltage (i.e., V_{min}).

C. Variable Width Links

Another way to reduce I/O power is to reduce the number of active I/O lanes; we refer to links that can vary its number of active lanes as *variable width links (VWL)*. VWLs are used in Infiniband and YARC switches [17] and are also

[2] Each unidirectional HMC link can be individually power controlled [5], like the unidirectional links in Infiniband and YARC switches [17]).

supported by HMCs [5]. While VWL reduces I/O power at the cost of reduced I/O bandwidth like DVFS, VWL provides distinct tradeoffs from DVFS. VWL does not incur the long SERDES latency overheads of DVFS; however, it provides less power reduction because it does not reduce dynamic energy per bit. When modeling VWL links, we allow the number of active lanes per link to be reduced from 16 down to 8, 4, or 1. We calculate the power when l lanes are on as $(l+1)/(16+1)$ of a full power link, as I/O clock power is similar to power of a lane [16]. We assume 1us latency for changing the number of active lanes [17].

V. Network-Unaware Management

Our first step in exploring how to utilize the above I/O power control mechanisms to reduce memory network power is to adapt prior works on memory power management to memory networks. Since no prior power management work has been previously proposed in the context of a memory network, we refer to our adaptation of prior works as *network-unaware power management*. We will explore network-aware power management in Section VI.

In this study, we explore hardware power management techniques that do not require software/OS assistance or cross-layer optimization. To manage VWL and DVFS links, we adapt [20], which uses hardware counters to simultaneously estimate the aggregate queuing and transmission latency overheads of read packets over a link for all possible link bandwidth configurations to adjust link bandwidth accordingly. To manage ROO links, we incorporate aspects of [21] and [22]; [21] describes a hardware mechanism to find the optimal power ROO mode for a given memory latency overhead constraint, while [22] hides wakeup latency overheads for responses from DRAM. Finally, to provide predictable worst-case performance overheads, we limit average memory latency overhead by incorporating feedback control and performance violation detection from [23].

Broadly, network-unaware management works as follows. It seeks to reduce power while keeping the aggregate memory read latency overhead of the network below an allowable memory slowdown or *AMS* in short; a memory network's AMS is set as a user-tunable factor of $\alpha\%$ times the network's aggregate memory read latency had all its links always operated in full power mode, and thus is given in units of time (as opposed to being an unit-less fraction). To this end, each memory module uses a hardware counter to track its aggregate DRAM read access latency. Each link controller contains a hardware counter to measure the link's actual aggregate read packet latency and also counters to estimate what the link's aggregate latency might be had the link always operated at full power. Using these values, each memory module independently determines the appropriate amount of AMS the module can have such that the network as a whole obeys the network-level AMS as determined by the user-settable factor α. Each module calculates its AMS periodically after each fixed time interval, referred to as an *epoch*; we assume 100us epochs, similar to [20]. After calculating its AMS, each module divides the AMS to among its various links; Section V-A details how to obtain the AMS of each link at the end of each epoch. Each link controller then sets its link to the lowest power mode whose latency overhead is less than its AMS. Section V-B describes how a link sets its power mode according to its AMS. Finally, each link controller periodically checks whether its current latency overhead exceeds its AMS [23]; if violation is detected, the link switches to full power until end of the epoch. Figure 10 summarizes the above.

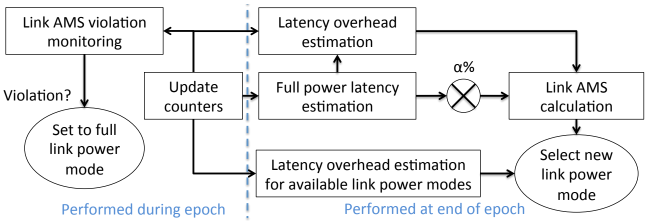
Link AMS violation monitoring
Violation?
Set to full link power mode
Update counters
Latency overhead estimation
Full power latency estimation
α%
Link AMS calculation
Latency overhead estimation for available link power modes
Select new link power mode
Performed during epoch
Performed at end of epoch

Figure 10. Network-unaware management overview.

A. Obtaining a Link's Allowable Memory Slowdown (AMS)

We refer to the estimated aggregate memory latency of a network or a module in an epoch had it operated in full power as the network or module's **f**ull power **e**poch **l**atency or *FEL*. Network-unaware management keeps aggregate memory latency overhead in an epoch within $\alpha\%$ times the network's FEL simply by requiring each module independently keep its latency overhead within $\alpha\%$ times the module's FEL. Specifically, network-unaware management calculates the network-level AMS for the next epoch as:

$$\begin{aligned} AMS_N(t{+}1) &= \alpha\%\cdot\textstyle\sum_0^m\sum_0^t FEL_{m,t} - \sum_0^m\sum_0^t (AEL_{m,t} - FEL_{m,t}) \\ &= \textstyle\sum_0^m [\alpha\%\cdot\sum_0^t FEL_{m,t} - \sum_0^t (AEL_{m,t} - FEL_{m,t})] \\ &= \textstyle\sum_0^m AMS_M(m,t{+}1) \qquad (1) \end{aligned}$$

Above, $FEL_{m,t}$ stands for the full power epoch latency of module m during epoch t; $AEL_{m,t}$ is the **a**ctual **e**poch **l**atency of m during epoch t, which is the actual measured aggregate latency of m throughout t; $AMS_m(m, t+1)$ is module m's AMS for the next epoch.

Module m obtains $AEL_{m,t}$ in Equation 1 by summing A) m's aggregate DRAM array read latency and B) aggregate link latency for read packets (i.e., read request and read response packets) during t. A) is the number of reads to the module's DRAM times DRAM access latency (e.g., 30ns from Table I). B) is the sum of the latency of all read packets passing through the links that connect the module upstream, which we refer to as the module's *connectivity links*; the link latency of each read packet is obtained as the difference of the departure time and the arrival time of the last flit of each read packet. Module m obtains $FEL_{m,t}$ in the same way as $AEL_{m,t}$ except that B) is estimated using a delay monitor and delay counter [20] per link configured to assume the link always operates at full power. At the end of each epoch, each module updates two additional hardware counters to record $\sum_0^t FEL_{m,t}$ and $\sum_0^t (AEL_{m,t} - FEL_{m,t})$, respectively. Finally, each connectivity link of m receives an equal portion of the module-level AMS.

B. Setting Link Power Mode according to the Link's AMS

After receiving its AMS, each link controller sets its link power mode for the next epoch as the minimum power mode whose predicted latency overhead during the next epoch is lower than or equal to the link's AMS; we refer to the predicted latency overhead of operating a link at a particular low power mode during the next epoch as the Future Latency Overhead or (*FLO*) of the given power mode of the given link; the FLO of a given link's given power mode is calculated as the estimated aggregate latency of operating the link at the given power mode during the current epoch minus the link's FEL during the current epoch.

We use the same method to estimate the FLO of VWL and DVFS power modes during an epoch since VWL and DVFS behave similarly. Each link uses a pair of hardware latency counters from [20] (referred to as the delay monitor and counter in [20]) to estimate the FLO of each available VWL/DVFS power mode of the link. For each DVFS low power mode, we also add to this estimated FLO the SERDES latency overhead of the power mode multiplied by the number of read packets over the link during the epoch.

We estimate the FLO latency for ROO power modes by adapting the idle interval histogram algorithm from [21]. The ROO power modes we evaluate have idleness thresholds of 32ns, 128ns, 512ns, and 2048ns, where each ROO power mode turns off a link after it has been idle for longer than the power mode's idleness threshold; the ROO power mode with 2048ns threshold is considered the full power mode (i.e., a ROO link always turns off after being idle for 2048ns). The idle interval histogram algorithm requires an idle interval bucket for every ROO mode. At the end of each link idle interval, the appropriate idle interval bucket is incremented. At the end of the epoch, the algorithm calculates the predicated latency overhead of a ROO mode by summing from the 32ns idle interval bucket to the bucket whose idle interval corresponds to that ROO mode, and then multiplying the sum by an estimated average latency overhead per wakeup. This average latency is wakeup latency + wakeup latency*the average number of read packet arrivals during wakeup; the latter number is estimated by periodically sampling how many subsequent read packets arrive during an amount of time equal to the wake up latency after a periodically chosen read packet first arrives.

We discovered in our experiments that waking up a request link can cause significant queuing latency overhead in a later response link because read response packets are much bigger (5X as many flits per packet, assuming 64B lines) than read request packets; for example, if a queue of N requests delayed in a waking request link all have the same destination module, when they eventually arrive at and then exit the DRAM array of the destination module, they can translate to a queue that is effectively five times as long at the destination module's response link. To account for any potential latency overhead that waking up a request link may inflict on other links, we add an additional value of wakeup latency*the average number of read packet arrivals during wakeup when estimating the predicated latency overhead of operating a request link under a given ROO power mode.

Finally, we estimate the FLO for VWL/DVFS +ROO links as the sum of the VWL/DVFS power mode's FLO and the ROO power mode's FLO.

C. Evaluation

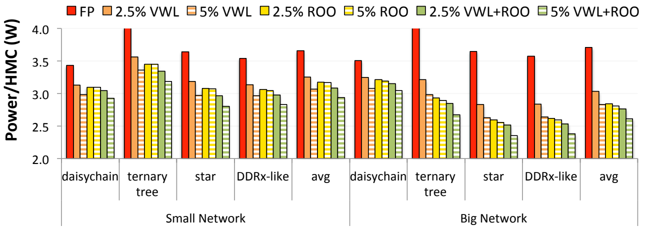

Figure 11. Per HMC power under network-unaware management.

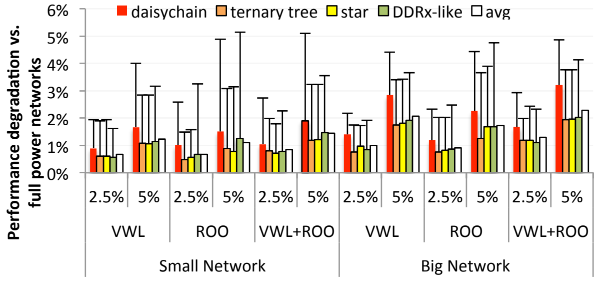

Figure 12. Average and maximum (error bar) performance overhead under network-unaware management.

Figure 11 shows the average power per HMC under network-unaware power management for VWL, ROO, and VWL+ROO links[3] and also average power for full power (FP) networks. On average across 336 comparisons (i.e., three circuit-level I/O power reduction mechanisms * two values of α * four topologies * 14 workloads), network-unaware management yields 14% average overall power reduction for small networks. Average power reduction increases to 24% for big networks; this is as expected since idle I/O power is higher in big networks (see Figure 5). The corresponding idle I/O power reduction is 32% and 21% for big and small networks, respectively. Figure 12

[3] We will evaluate DVFS and DVFS+ROO links in Section VI-D.

shows that the throughput[4] overhead of network-unaware power management. The maximum throughput degradation for $\alpha = 2.5\%$ and $\alpha = 5\%$ are 3.2% and 5.1%, respectively, which closely follow their respective α. This shows the effectiveness of using memory latency overhead feedback control to curb overall system performance overheads; the occasional performance degradation in excess of α is due to imperfect link latency overhead estimations using counters.

Memory power management provides star and the DDRx-like topologies with the highest power reduction relative to total network power. These topologies allow many modules with cold memory ranges (see the flat line segments in Figure 4) to go into very low link power modes; under daisy chain, however, such modules still need to be frequently traversed to provide access to modules with hot memory content. Ternary tree, on the other hand, contains entirely of high radix HMCs, with high logic and DRAM leakage power (see Figure 5); as such, idle I/O power reduction as a fraction of total network power is less pronounced.

Overall, idle I/O power still consumes 44% and 57% of total power for the small and big networks, respectively, and thus remains the top power contributor. One way to further reduce idle I/O power is to increase the network's AMS by increasing α. Figure 11 shows that power reduction through increasing α is very modest, only 3% on average when $\alpha\%$ goes from 2.5% to 5%; meanwhile, Figure 12 shows that the average throughput degradation almost doubled from 0.9% to 1.7%. While 0.9% to 1.7% average system-level performance overheads may be acceptably small, they are not negligible. Further increasing α only further degrades system-level performance for modest gains in power reduction. As such, new techniques need to be explored for further power reduction while minimizing performance overheads.

VI. Network-Aware Management

One main problem with network-unaware management is that busier links often operate in a lower power mode than links with lower utilization. The left half of Figure 13 plots the fraction of total link hours (i.e., analogous to machine hours) spent by links of different utilization levels for VWL links under network-unaware management for big networks; 9% of total link hours are spent by links with 0-1% utilization in 16-lane mode (i.e., the red segment in "0-1%" bar for big networks); meanwhile, roughly the same total amount of link hours are spent by links with 5+% utilization in 8-lane mode (i.e., the combined height of the orange segments in the last three bars for big networks). Intuitively, a busier link should operate in a higher power mode since it incurs more frequent and, therefore, higher total latency overhead than a lower utilization link for operating at the same low power mode. The counter-intuitive behavior is because a more frequently accessed module often generates more AMS than a less frequently accessed link; unfortunately, under network-unaware management, the large amount of AMS generated by a frequently-accessed module is also assigned to that module, allowing it to often operate at equal or lower power modes than infrequently accessed links. Ideally, one would like to see a link hour distribution like one shown in the right half of Figure 13 that increases the time low utilization (e.g., 0-5% utilization) links spend in low power modes by decreasing the time high utilization (10+% utilization) links spend in low power modes; network-aware power management obtains this distribution.

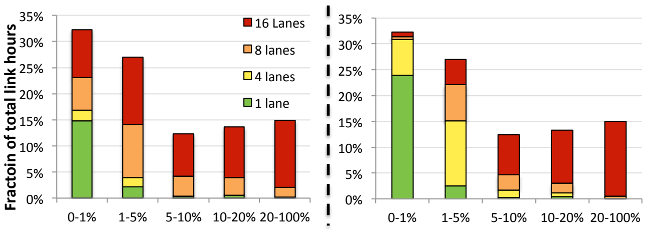

Figure 13. Left: Distribution of link hours spent in different VWL modes (y-axis) by links of different utilizations (x-axis) under network-unaware management. Right: Distribution under network-aware management.

Memory networks also provide new opportunities to reduce link power at low performance cost. We observe that for ROO links, multiple links along the access path can wake up at the same time, instead of one at a time, to enable aggressive ROO modes while minimizing wake up overheads. We also observe that latency overheads at a downstream link does not cause memory latency overhead if an upstream response link is congested; had there been no delay downstream, the packet would arrive at the congested upstream response link sooner and wait longer in queue.

Network-aware management addresses and exploits the above problems and opportunities. It builds on top of network-unaware management: it also relies on Equation 1 to calculate network-level AMS and uses the same hardware link counters for FLO estimation, etc. The difference is that instead of each module independently obtaining its AMS, network-aware management intelligently redistributes the network-level AMS across the network to ensure that busier links operate at no lower power modes than less busy links; this is described in Section VI-A. Network-aware management also completely hides the wakeup latency of response links, and aggressively sets downstream response links to lower VWL/DVFS modes when upstream response links are congested, as will be described in Section VI-B and Section VI-C, respectively.

A. *Network-Aware Slowdown Redistribution*

We propose redistributing AMS across the network such that busier links always operate at higher or equal power mode than less busy links. This allows the leftover AMS

[4] Since every workload contains multi-threaded application(s), we use FLOPS for workloads with floating point applications only and use memory accesses per second for the rest (seven total).

in the former to be transferred to the latter to enable the latter to operate in lower power modes. There are two challenges with network-aware slowdown redistribution. First, determining the relative utilization of different links can be challenging since link utilization can change even within an epoch. Second, even when the relative utilization of different links is known, there are still numerous ways to set link power modes across the network such that busier links also have have higher or equal power modes; an efficient and effective selection method needs to be identified.

To address the first challenge, we observe that memory traffic attenuates across the network from memory modules closer to the processor to memory modules farther away from the processor. This implies that among links of the same type (i.e., request or response link), an upstream link always have equal or higher utilization than its immediate downstream link. To address the second challenge, we observe that a distributed algorithm can exploit the fact that each module is aware of its neighbors in a memory network to make topology-aware power mode decisions without having to first translate the physical topology into a logical data structure. A distributed algorithm also divides the computational and memory overheads over all modules, resulting in low overheads per module.

Exploiting the above observations, we propose Iterative Slowdown Propagation (ISP), a distributed message passing algorithm that distributes AMS over the network through several iterations (our evaluations cap total iterations at three). Each iteration consists of two steps - scatter and gather. ISP scatter redistributes unused AMS; ISP gather collects unused AMS and other helpful statistics and enforces that an upstream link always be set to higher or equal power mode than downstream links of the same type. Figure 14 illustrates the direction of message passing for these two ISP steps. The final CLS (see Section V-A) at each link by the end of ISP is used to select the power mode.

1) ISP Scatter: ISP scatter broadly works as follows. At the i^{th} ISP iteration, only some links may still benefit from receiving more AMS; we refer to a link previously determined to potentially benefit from receiving more AMS as a *slowdown receiving candidate (SRC)*. Each link L divides its unused AMS equally among all downstream SRCs, or *DSRC*s, of the same type as L; since the latency overhead of a low power mode is usually higher for busier links, equal AMS distribution helps to converge to a global link power mode selection where busier links select no lower power modes. We describe ISP scatter in detail below.

At the beginning of ISP scatter, the head module contains the total network-wide unused AMS (which is obtained by ISP gather of the previous iteration, to be described in Section VI-A2). The head module divides the unused AMS by the total SRCs in the network to calculate a per candidate slowdown (*PCS*) value and passes the PCS to the link controllers of the head module's connectivity links.

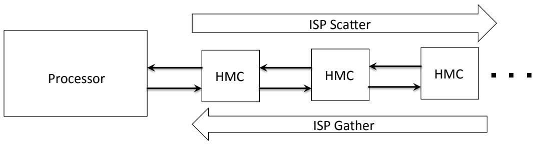

Figure 14. ISP message flow. Each HMC passes the same message packet to each of its downstream/upstream neighbor(s) during ISP scatter/gather.

If a link is not an SRC, upon receiving a PCS message, the link controller simply passes the received PCS value to all downstream links of the same type.

If a link is an SRC, upon receiving a PCS message, the link controller increases the link's AMS by the received PCS and selects (but not yet physically set) the lowest link power mode whose FLO is below the updated AMS. Next, the link controller passes $PCS + (AMS - FLO)/DSRC$ as the new PCS value to each immediate downstream link of the same type; this effectively evenly distributes to these downstream links any leftover AMS at an upstream link after the upstream link selects its power mode. Lastly, the link controller updates the link's AMS as the FLO of the selected power mode and also decides whether the link should be an SRC during the next iteration of ISP scatter; it decides true if the link has not already selected the lowest available power mode and if $PCS + AMS$ is at least a big fraction (e.g., 25%) of the next lower power mode's FLO.

2) ISP Gather: ISP gather obtains for every link controller its DSRC value and also obtains for the head module the network-wide unused AMS via parallel reduction by passing messages upstream. The DSRC values are obtained via a parallel prefix sum reduction over the network in which each leaf link controller passes '1' if it is an SRC, '0' otherwise, to the immediate upstream link controller of the same type. Each non-leaf link controller accumulates the values in the received messages, sets DSRC to this partial sum, increments the partial sum by '1' if the link is itself an SRC, and passes the updated sum upstream similarly.

To obtain the network-wide unused AMS, the first ISP gather iteration calculates the initial network-wide AMS generated during the previous epoch, while each subsequent iteration calculates how much AMS is still unused after the previous ISP scatter iteration. The head module is responsible for calculating via Equation 1 the network-wide AMS generated during the previous epoch and for keeping track of the $\sum_0^m \sum_0^t FEL_{m,t}$ and $\sum_0^m \sum_0^t (AEL_{m,t} - FEL_{m,t})$ sums in Equation 1. To update these sums using the $AEL_{m,t}$ and $FEL_{m,t}$ values of the current epoch, $\sum_0^m FEL_{m,t}$ and $\sum_0^m (AEL_{m,t} - FEL_{m,t})$ are obtained via a simple parallel reduction sum operations over the $AEL_{m,t}$ and $FEL_{m,t}$ values, respectively, across all modules in the network. For subsequent iterations, ISP gather accumulates the unused AMS in the leaf modules also by a simple parallel reduction sum across the network.

Finally, to ensure that an upstream link L's power mode will be equal or higher than the maximum power mode among all L's downstream links of the same type, each downstream link also passes its currently selected power mode to its upstream link of the same type during ISP gather. If L's selected power mode is lower than any of its downstream links, L increases its power mode to match the latter, updates CLS and passes upstream the difference between the FLOs of the updated and the previous power mode selection as unused AMS. Note that since the total amount of data passed from a downstream module to its upstream module during ISP gather is small, each module only needs to send a single 64B packet during ISP gather.

3) Utilizing the Leftover AMS from ISP: After the end of the last iteration of ISP gather, all leftover AMS in the network is stored in the head module. Network-aware management utilizes this unused AMS to prevent some links from switching to full power mode due to AMS violation during an epoch. When detecting a violation, a link controller first requests for some of the unused AMS recorded at the head module. When receiving the AMS request message, the head module responds with a portion of the unused AMS (e.g., $1/16^{th}$ of the original unused AMS), if the unused AMS has not already been depleted by AMS request messages prior in the epoch. We allow each link to request up to a quarter of the original unused AMS (e.g., by allowing a maximum of $0.25/(1/16) = 4$ AMS requests per link per epoch). A link switches to full power mode if the AMS request is denied.

B. Network-Aware Optimizations for ROO

Network-unaware management adapts [22] to reduce wake up latency for response links (see Section V); while the DRAM array of a module is still being accessed, the module proactively wakes up its response link (if the link is off), instead of first waiting for the DRAM access to complete, to reduce the performance overhead of waking up the response link. Since the DRAM access latency is typically longer than link wakeup latency (e.g., 30ns vs. 14ns) , the wake up latency overhead of the response link of the module being accessed can be completely hidden.

Network-aware management seeks to not only hide the response link wakeup latency of *the module under access*, but also hide the wakeup latency of *every response link along the response path to the processor*. It does so by again ensuring that a downstream link always be in a higher or equal power mode than its downstream link(s). Specifically, under network-aware management, a response link starts turning on either when the link's module's DRAM array is being read or after one of its immediate downstream response links starts turning on plus a wait interval; the interval is the sum of router latency and the downstream link's current SERDES and transmission latencies, which are constant values during an epoch. A response link only turns off when the link's DRAM is not being read and when all immediate downstream response links are off, which implies that the latter links also will not be soon receiving packets from their respective downstream response links or their modules' DRAMs. Note that an upstream response link's transmitter and its immediate downstream response links' receivers all reside on the same module; as such, the on/off state of immediate downstream links are readily available.

Since network-aware management completely hides wakeup latency overheads for response links, it only considers the request links to be SRCs during ISP for networks with only ROO links. For the similar reason, for VWL/DVFS + ROO links, the head module assigns during ISP scatter more (i.e., $3/4^{th}$ of) unused AMS to the request links.

C. Network-aware Optimizations for VWL/DVFS

When upstream response links are congested, network-aware management ignores some of the latency overhead experienced by downstream links when calculating of network-wide AMS at the end of an epoch. To do so, each response link controller tracks for the current epoch the cumulative queuing delay (QD) and the fraction of packets that are queued, or simply *queuing fraction (QF)*. A packet is conservatively considered as being queued if it arrives behind at least three older packets according to the link's full power mode delay monitor. The reduction sum operation of first ISP gather iteration, which calculates how much network-wide AMS is available for the next epoch, makes use of these statistics; during this operation, each response link controller reduces the total downstream overhead (from both downstream request and response links) by the minimum of $downstream\ overhead * QF$ and QD.

D. Evaluation

Figure 15 shows the network-wide power reduction of network-aware management vs. network-unaware management. Overall power reduction over network-unaware power management is, on average, 11% and 19% for small and big networks, respectively. The corresponding I/O power reduction for small and big networks are 17% and 29%, respectively. Figure 16 presents the benchmark-level power reduction of network-aware and unaware management vs. full power networks, on average across all topologies; for

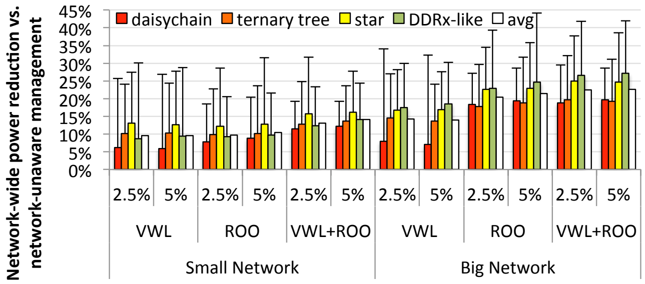

Figure 15. Power savings of network-aware vs. unaware management.

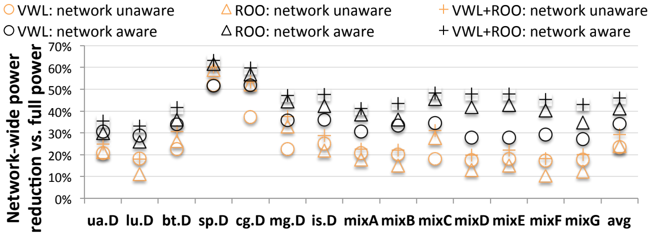

Figure 16. Power saving by workload for big networks.

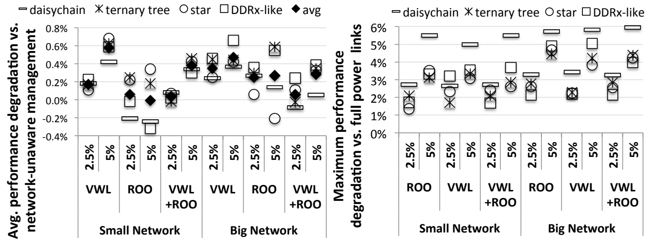

Figure 17. Left: Average performance overhead vs. network-unaware management. Right: Maximum performance overhead vs. full power networks.

readability, it only reports big networks and $\alpha = 5\%$. Figure 16 shows that network-aware management consistently yields higher power reduction for every workload.

The left half of Figure 17 shows the average performance overhead of network-aware management vs. unaware management. For $\alpha = 2.5\%$ and $\alpha = 5\%$, network-aware management incurs a 0.2% and 0.3% average performance penalty, respectively, compared to the latter. Closer inspection reveals that some high utilization links under network-unaware management do not make use of any of their AMS because the latency overheads of putting these links to low power mode are exceedingly high. Under network-aware power management, the AMS generated by these links is redistributed to other links that can utilize the AMS, resulting in a corresponding performance degradation. The right half of Figure 17 shows the maximum performance overhead of network-aware management vs. full power. The maximum overhead over all 672 comparisons is 5.9%.

We perform sensitivity analysis by evaluating DVFS links instead of VWL links and evaluating ROO with a wakeup latency of 20ns instead of 14ns. Figure 18 shows the network-wide power reduction and performance overheads of network-aware and network-unaware management relative to full power networks for $\alpha = 5\%$. Under DVFS, both schemes yield less power reduction for the same value of $\alpha\%$ (e.g., $< 5\%$) than VWL; this is due to the high SERDES latency overheads at low voltage. As expected, the power savings under both schemes for the longer 20ns ROO links are slightly reduced. Meanwhile, network-aware power management provides 21% and 12% power reduction over vs. network-unaware power management for big and small networks, respectively, on average across DVFS, 20ns ROO, and DVFS+20ns ROO links.

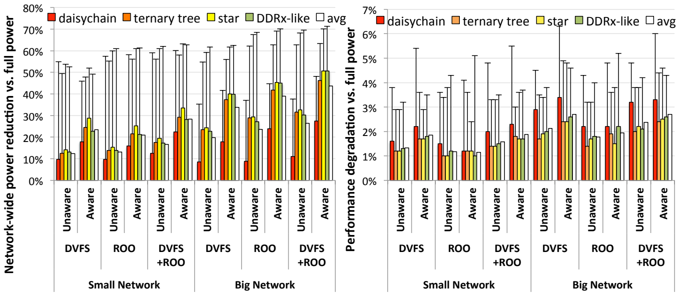

Figure 18. Left: Average and maximum power reduction for DVFS and 20ns ROO links. Right: Average and maximum performance overheads.

VII. Discussion

A. Other I/O Power Reduction Strategies

An alternative approach for selecting link bandwidth (but not ROO modes) is to statically reduce link bandwidth like the well-known fat-trees and tapered-trees. We note that if traffic is evenly distributed across the network (e.g., by interleaving adjacent pages across all modules), a hybrid fat-tree and tapered-tree static bandwidth selection[5] does not induce any queuing latency overheads. However, low bandwidth links still takes longer to transmit a packet; since static selection cannot control the aggregate packet transmission latency overheads, it only provides a static power and performance tradeoff point and incurs unpredictable worst-case performance overheads. For example, for the VWL power/performance model and big networks, static selection+interleaving incurs 13% average performance overheads, 43% worst-case overhead, and 30% average top quarter worst-case overheads, over the four topologies and 14 workloads (i.e., $4 * 14 = 56$ comparisons).

In comparison, our network-aware management not only provides tunable design points, but also provides higher benefit for the same performance. By sweeping α values, we found that network-aware power management with $\alpha = 30\%$ matches the average performance overhead above, but reduces overall power by 15% compared to static selection; this is because our network-aware power management allows contiguous memory pages to be mapped within the same HMC, which consolidates accesses to fewer active HMCs and allows more HMCs to go into low power modes. In addition, network-aware management only incurs a 25% worst-case and 20% average top quarter worst-case performance overheads vs. full power networks.

[5] Let $S(x)$ be the number of links with hop distance x and T be the total number of links, a fat+tapered tree sets the bandwidth of link with hop distance d as $1/S(d) * (1 - sum_1^{d-1} S(i)/T)$ of maximum bandwidth (and raises it to the nearest available bandwidth option).

B. Related Work

Many prior works have studied power management for on-chip networks. Nodes (i.e., cores) in on-chip networks need to communicate with one another and, therefore, benefit from a small average distance between the network nodes; as such, on-chip networks use topologies with many redundant links, such as the well-known 2D mesh, which provide very different power management challenges and opportunities from the minimally connected topologies we study.

VIII. Conclusion

In this paper, we perform the first exploration to understand the power characteristics of memory networks. We identify idle I/O power as the highest power contributor. Subsequently, we study idle I/O power in more detail. We evaluate well-known I/O power reduction techniques such as DVFS, ROO, and VWL. We adapt existing works on memory power management to memory networks and obtain 32% and 21% I/O power reduction for big and small memory networks, respectively. Finally, we explore network-aware management and reduce I/O power by another 29% and 17% for big and small networks, respectively.

Acknowledgment

This work was supported in part by NSF, Cisco, and CFAR, within STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA. We also thank Robert Patti, Mrunmay Talegaonkar, and Guanghua Shu for their feedback at different stages of this work.

References

[1] J. Zerbe, P. Chau, C. Werner, T. Thrush, D. Perino, B. Garlepp, and K. Donnelly, “1.6 gb/s/pin 4-pam signaling and circuits for a multi-drop bus,” in *VLSI Circuits.*, 2010.

[2] B. Ganesh, A. Jaleel, D. Wang, and B. Jacob, “Fully-buffered dimm memory architectures: Understanding mechanisms, overheads and scaling,” in *HPCA*, 2007.

[3] D. Resnick, “Memory network methods, apparatus, and systems,” 2010. [Online]. Available: http://www.google.com/patents/US20100211721

[4] G. Kim, M. Lee, J. Jeong, and J. Kim, “Multi-gpu system design with memory networks,” in *MICRO*, 2014.

[5] “Hybrid Memory Cube Specification 2.1,” 2015, http://www.hybridmemorycube.org/specification-v2-download-form/.

[6] A. N. Udipi, N. Muralimanohar, N. Chatterjee, R. Balasubramonian, A. Davis, and N. P. Jouppi, “Rethinking dram design and organization for energy-constrained multi-cores,” in *ISCA*, 2010.

[7] B. Giridhar, M. Cieslak, D. Duggal, R. Dreslinski, H. M. Chen, R. Patti, B. Hold, C. Chakrabarti, T. Mudge, and D. Blaauw, “Exploring dram organizations for energy-efficient and resilient exascale memories,” in *SC*, 2013.

[8] B. M. Rogers, A. Krishna, G. B. Bell, K. Vu, X. Jiang, and Y. Solihin, “Scaling the bandwidth wall: Challenges in and avenues for cmp scaling,” in *ISCA*, 2009.

[9] E. Cooper-Balis, P. Rosenfeld, and B. Jacob, “Buffer-on-board memory systems,” in *ISCA*, 2012.

[10] MICRON, “4Gb:x4, x8, x16 DDR4 SDRAM,” https://www.micron.com/~/media/documents/products/data-sheet/dram/ddr4/4gb_ddr4_sdram.pdf.

[11] J. T. Pawlowski, “Hybrid Memory Cube (HMC),” *Hot Chips 23*, 2011.

[12] S. Pugsley, J. Jestes, R. Balasubramonian, V. Srinivasan, A. Buyuktosunoglu, A. Davis, and F. Li, “Comparing implementations of near-data computing with in-memory mapreduce workloads,” *Micro, IEEE*, 2014.

[13] S. P. Muralidhara, L. Subramanian, O. Mutlu, M. Kandemir, and T. Moscibroda, “Reducing memory interference in multicore systems via application-aware memory channel partitioning,” in *MICRO*, 2011.

[14] J. Jeddeloh and B. Keeth, “Hybrid memory cube new dram architecture increases density and performance,” in *VLSI Technology*, 2012.

[15] K. T. Malladi, B. C. Lee, F. A. Nothaft, C. Kozyrakis, K. Periyathambi, and M. Horowitz, “Towards energy-proportional datacenter memory with mobile dram,” in *ISCA*, 2012.

[16] G. Shu, W. S. Choi, S. Saxena, S. J. Kim, M. Talegaonkar, R. Nandwana, A. Elkholy, D. Wei, T. Nandi, and P. K. Hanumolu, “23.1 a 16mb/s-to-8gb/s 14.1-to-5.9pj/b source synchronous transceiver using dvfs and rapid on/off in 65nm cmos,” in *International Solid-State Circuits Conference*, 2016.

[17] D. Abts, M. R. Marty, P. M. Wells, P. Klausler, and H. Liu, “Energy proportional datacenter networks,” in *ISCA*, 2010.

[18] T. Anand, M. Talegaonkar, A. Elkholy, S. Saxena, A. Elshazly, and P. Hanumolu, “3.7 a 7gb/s rapid on/off embedded-clock serial-link transceiver with 20ns power-on time, 740 uw off-state power for energy-proportional links in 65nm cmos,” in *International Solid-State Circuits Conference*, 2015.

[19] T. M. Andersen, F. Krismer, J. W. Kolar, T. Toifl, C. Menolfi, L. Kull, T. Morf, M. Kossel, M. Brndli, P. Buchmann, and P. A. Francese, “4.7 a sub-ns response on-chip switched-capacitor dc-dc voltage regulator delivering 3.7w/mm2 at 90in *International Solid-State Circuits Conference*, 2014.

[20] J. Ahn, S. Yoo, and K. Choi, “Dynamic power management of off-chip links for hybrid memory cubes,” in *DAC*, 2014.

[21] D. Wu, B. He, X. Tang, J. Xu, and M. Guo, “Ramzzz: Rank-aware dram power management with dynamic migrations and demotions,” in *SC*, 2012.

[22] K. Malladi, I. Shaeffer, L. Gopalakrishnan, D. Lo, B. Lee, and M. Horowitz, “Rethinking dram power modes for energy proportionality,” in *MICRO*, 2012.

[23] X. Li, Z. Li, Y. Zhou, and S. Adve, “Performance directed energy management for main memory and disks,” *Trans. Storage*, 2005.