Modeling Communication Costs in Blade Servers

Qiuyun Wang, Benjamin C. Lee
Duke University
Department of Electrical and Computer Engineering
{qiuyun.wang, benjamin.c.lee}@duke.edu

ABSTRACT

Datacenters demand big memory servers for big data. For blade servers, which disaggregate memory across multiple blades, we derive technology and architectural models to estimate communication delay and energy. These models permit new case studies in refusal scheduling to mitigate NUMA and improve the energy efficiency of data movement. Preliminary results show that our model helps researchers coordinate NUMA mitigation and queueing dynamics. We find that judiciously permitting NUMA reduces queueing time, benefiting throughput, latency and energy efficiency for datacenter workloads like Spark. These findings highlight blade servers' strengths and opportunities when building distributed shared memory machines for data analytics.

Categories and Subject Descriptors

C.0 [**Computer Systems Organization**]: General—*System architectures*; C.4 [**Computer Systems Organization**]: Performance of Systems—*Design studies, Modeling techniques*; B.3 [**Hardware**]: Memory Structures—*Shared memory*

Keywords

NUMA, technology models, energy and delay, blade servers, scheduling, communication cost

1. INTRODUCTION

Blade servers provision abundant memory in a dense form factor and their distributed shared memory are well suited to big data applications. Researchers have prototyped or emulated blade architectures [11, 6] to understand their potential and to demonstrate key capabilities, such as fine-grained access and address translation. Beyond specific designs, however, researchers must assess sensitivity to technology parameters and explore server organizations. Unfortunately, existing experimental methods lack the required flexibility.

HotPower'15 October 04-07, 2015, Monterey, CA, USA

DOI: http://dx.doi.org/10.1145/2818613.2818743.

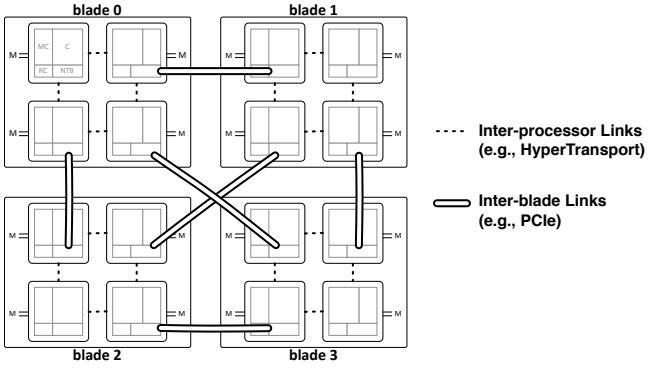

Figure 1: A representative blade architecture with sixteen-core processors, four-processor blades, and a four-blade server.

In this paper, we present technology models that enumerate communication paths through a blade server and identify contributors to delay and energy. In addition to DRAM costs, we account for inter-processor and inter-blade data transfers. With these models, researchers can explore scenarios in system organization and data movement, identifying opportunities and addressing challenges.

One particular challenge is non-uniform memory access (NUMA). In a blade server, a processor can retrieve data from memory via several communication paths that introduce multiple levels of NUMA. Conventional wisdom argues that NUMA harms performance and should be avoided [19, 2]; prior work has proposed task and data managers that restrict remote memory access [12, 9, 17, 3]. However, NUMA might improve performance; a system that selectively permits NUMA might dequeue tasks sooner and increase throughput. Technology models help researchers understand the benefits of permitting NUMA and develop new schedulers that balance latency and throughput.

In §2, we present technology models that enumerate communication paths through a blade server. We deploy this model for server simulation in two case studies, scheduling to mitigate NUMA and showing the efficiency of remote data access over data migration (§3–§5).

2. MODELING COMMUNICATION COST

We derive cost estimates for three types of communication channels in a blade architecture (e.g., Figure 1): memory bus, inter-processor links, inter-blade links. By simulating a server with these estimates, we assess a task's communication efficiency. Table 1 summarizes our estimates of communication costs.

Technology	DDR3 8 2Gb,x8	HT3.1 PTP	PCIe 2.0 PTP
Data Rate (GB/s per lane)	0.2	0.8	0.5
Lanes	x64	x32	x16
Uni-directional B/W (GB/s)	12.8	25.6	8.0
Transfer Energy (pJ/bit)	160.0 (@20% util.) 70.0 (@100% util.)	36.0	37.5
Latency (ns)	50-100	100 RT	190 RT

Table 1: Summary of technology models and estimates

Memory Communication. The memory controller is integrated into the processor die and responds to last-level cache misses. Memory latency is affected by array latency, queueing delays, and request scheduling. The controller can schedule memory requests for a mix of latency, throughput, and fairness targets. Sophisticated scheduling mitigates row buffer misses and bank conflicts, which add to memory latency.

DRAMs consume dynamic energy due to precharges, activates, reads, and writes. In addition, DDRx interfaces, which include delay-locked loops and on-die termination, draw static current regardless of channel utilization. Thus, high-performance memories are energy-disproportional for tasks with modest bandwidth demands [13]. 2Gb DDR3 with 8DQs strikes a balance between chip capacity and I/O width [13] and dissipate approximately 2W per GB. A typical server has 8GB per channel, 2 channels per processor, and 4 processors per blade. In total, 64GB of DRAM dissipates 128W.

Inter-Processor Communication. Multiple processors are integrated into a blade. They share a physical address space and can support coherent access to shared memory. Both the local memory controller and the interconnect controller observe a last-level cache miss. The latter uses the memory address to identify and route a memory request destined for another processor's controller.

A remote memory request may require one or two hops, which introduces latency and energy costs in addition to those from DRAM. The interconnect uses serial, point-to-point links for high data rate and low latency. We use HyperTransport (HT) for the inter-processor connection [5]. HT transmitter logic requires 18ns to encode contents and add headers. Packet transmission and link interface circuitry adds 14ns. Receiver logic requires another 18ns. In total, the packetized request requires 50ns to reach a remote memory controller. With round-trip overheads and DRAM access delay, remote data access requires at least 150ns.

To assess energy costs, we examine serial links and their interfaces. A serial link requires serializer/deserializer (SerDes) circuitry at its endpoints to convert data between parallel and serial interfaces. This circuitry determines the energy cost per bit transferred. How this cost translates into system power depends on the number of processors, the number of links between them, and the data rates of those links [4].

HT links connect two processors with 16 lanes, each implemented with paired serial links for differential signaling. Bi-directional communication requires two paired links since a serial link is unidirectional. Thus, 16 lanes requires 64 links and 128 SerDes. Each SerDes consumes 10pJ per bit transferred and can transfer up to 6.4Gb per second [7]. We estimate power by multiplying the number of interfaces, the transfer rate and the cost per transfer – power is 8.2W per path and a server with five paths dissipates up to 40W.

Inter-Blade Communication. Blades share an address space and communicate via a backplane interconnect. Bridges and switches perform address translation and route memory requests to the appropriate blade [6, 18]. We describe a blade architecture in which a processor can access a remote blade's memory with load/store instructions via PCIe. The memory controller, PCIe root complex, and non-transparent bridges (NTBs) are all integrated into the processor die [6]. Each inter-blade connection begins and ends with an NTB bridge or switch. A typical 64b PCIe transmission requires 240ns of which 50ns is attributed to DRAM [5]; the remaining 190ns is the round trip PCIe link transmission delay.

Inter-blade communication also depends on the number of HT hops. For example, six PCIe links connect twelve NTBs in Figure 1. A memory request may traverse HT to reach the correct bridge and traverse PCIe to reach the correct blade. We calculate the expected HT delays on sending and receiving blades, incurred in addition to PCIe and DRAM latencies. With round-trip overheads, accessing data on a remote blade may require 410ns, including 50ns for DRAM, 190ns for PCIe, and 170ns for expected HT delay.

Energy costs increase with blade connectivity. In our example, each uni-directional connection is 16 lanes-wide and a bi-directional connection requires 32 lanes that dissipate 4.8W. Each lane dissipates 150mW and transfers data at 0.5GB/s [8]; energy cost is 37.5pJ/bit. Bridge logic and SerDes circuitry dissipate an additional 2.5W. In total, an inter-blade link dissipates 10W for 32 lanes and two bridges.

We consider connectivity between 4 blades, each with 4 processors and integrated bridges. These NTBs support back-to-back PCIe connections between pairs of blades. A fully-connected network requires 6 connections and 12 NTBs, which in total dissipate up to 60W.

In summary, our technology models are accurate and consistent with prior measurements. **Memory** estimates use DRAM specifications [15]. **Inter-Processor** estimates use HT technology parameters [5] to calculate latency from packet processing and transmission. Estimates align with NUMA measurements [2]. Inter-processor power numbers (40W at peak and 10-20W at typical utilization) are consistent with industry measurements [1]. **Inter-Blade** estimates are consistent with emulation parameters for disaggregated memory [10]. Power estimates are derived from PCIe bridge and link specifications [16, 7].

3. CASE STUDIES IN MANAGING NUMA

We illustrate opportunities for parameterized design and management with two case studies that deploy our technology models. First, we adapt queueing policies to the costs of varied communication paths. Second, we compare the energy efficiency of remote execution against migration.

Execution Model. Run-time systems rely on task queues to produce parallelism while preserving a programming model's clean abstractions (e.g., MapReduce, Spark, GraphLab). The task at the head of the queue is likely to find its data already in main memory because of caching or pre-fetching.

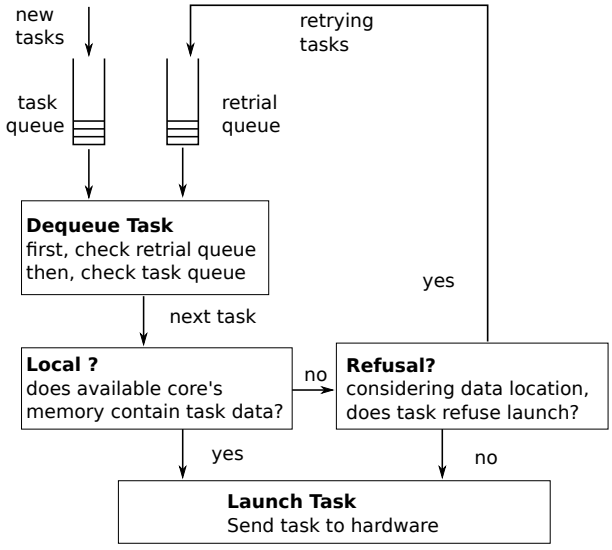

Figure 2: Refusal policies and queue management.

For example, Spark caches data from the current iteration of a machine learning kernel to ensure its availability for the next one.

However, non-deterministic queueing complicates the coordination of task scheduling and data placement; the data placement mechanism cannot predict which core will become available at a certain time. A lack of coordination exposes NUMA in blade servers. Tasks must navigate multiple levels of NUMA and still guarantee service quality. We look beyond the bimodal locality classification (e.g., local versus remote) and examine distance to data when scheduling tasks with NUMA.

Refusal Scheduling. We draw inspiration from delay scheduling [20], which improves storage locality for MapReduce tasks. Our case studies present policies that dictate whether a queued task should refuse execution on an available core due to NUMA.

Figure 2 illustrates refusal scheduling. Arriving tasks enter a queue. When a core becomes available, the scheduler determines this core's proximity to data required by the next task (e.g., address of task's vertex given an graph in distributed memory). The scheduler's refusal policy selectively refuses and permits NUMA. Refusing tasks enter a high-priority retrial queue. Queues prioritize tasks with earlier arrival times for FIFO fairness. A refusal limit is used to avoid starvation. We consider four refusal policies for multiple NUMA levels (e.g., Figure 1).

- **Local Execution (Local).** Task runs on processor for which its data is local. Otherwise, task refuses.
- **Inter-Processor 1-Hop Execution (IP-1).** Task favors local execution. It also accepts 1-hop inter-processor communication.
- **Inter-processor 2-Hop Execution (IP-2).** Task prefers running on blade for which its data is local. It refuses inter-blade communication.
- **Inter-blade Execution (IB).** Task executes on any available core. It favors cores closer to its data.

A refusal policy balances queueing and service time. The optimal policy depends on distance to data, sensitivity to NUMA, server utilization, as well as performance and efficiency goals.

#	Name	Bandwidth MI/CI,MB/s	Penalty HT-1	Penalty PCIe
Apache Spark				
S1	Word count	MI	264	1.17x	1.67x
S2	Logistic regression	MI	242	1.17x	1.60x
S3	Pagerank	MI	276	1.22x	1.76x
S4	Transitive closure	MI	235	1.19x	1.62x
S5	Alternating least squares	MI	231	1.16x	1.55x
S6	K-means	MI	283	1.22x	1.76x
S7	Pi	CI	21	1.01x	1.04x
Phoenix MapReduce				
M8	Word count	MI	111	1.10x	1.33x
M9	Histogram	CI	42	1.02x	1.05x
M10	String match	CI	28	1.04x	1.12x
M11	Linear regression	CI	7	1.0x	1.0x
Parsec				
P12	Facesim	EMI	2159	1.39x	2.46x
P13	Dedup	CI	50	1.09x	1.19x
P14	Bodytrack	CI	35	1.02x	1.06x
P15	Vips	CI	23	1.01x	1.04x
P16	Ferret	CI	73	1.04x	1.18x
P17	Raytrace	CI	19	1.02x	1.06x
P18	Swaptions	CI	5	1.00x	1.00x
P19	Streamclusters	CI	4	1.00x	1.00x
P20	Blackscholes	CI	3	1.00x	1.00x

Table 2: Benchmarks, their demand for remote memory bandwidth (Extremely Memory-Intensive, Memory Intensive, Compute Intensive) and latency penalties from NUMA.

4. EXPERIMENTAL METHODOLOGY

We link multiple simulators to coordinate the study of NUMA and queueing dynamics. Processor/memory simulation quantify task sensitivity to NUMA. Queueing simulation quantifies system latency and throughput.

Processor Simulation. We use Marssx86 and DRAMSim2 for processor and memory respectively. We simulate a 4-way OoO core (2GHz) with 128KB iCache and dCache, 2M L2 cache, and DDR3 with a 667MHz bus. Each benchmark is simulated on the region of interest for 200 million instructions; the results are shown in Table 2. When accessing remote memory, we add inter-processor and inter-blade latencies according to the technology models in §2. We focus on communication for heap data, assume that local memory holds code and stack. The simulator identifies address regions for code, stack, and heap to distinguish remote requests from local ones.

Queueing Simulation. We implement a discrete event simulator, modeled after BigHouse [14], to evaluate scheduling policies and queueing dynamics within a blade server (e.g., Figure 1). The queueing simulator uses service times from processor simulations, which account for NUMA, to estimate response times in G/G/k queues. It tracks each task and reports summary statistics for service quality (e.g., 95th percentile response time). We implement the **Local**, **IP-1**, **IP-2** and **IB** policies and simulate them for hundreds of millions of tasks. We assume the data distribution and demand is uniformly and randomly distributed among a blade server's processors.

Benchmarks. Table 2 summarizes tasks' bandwidth demands, which correlates with NUMA-induced performance penalties. EMI tasks must execute on cores with data in local memory; these tasks would otherwise saturate inter-

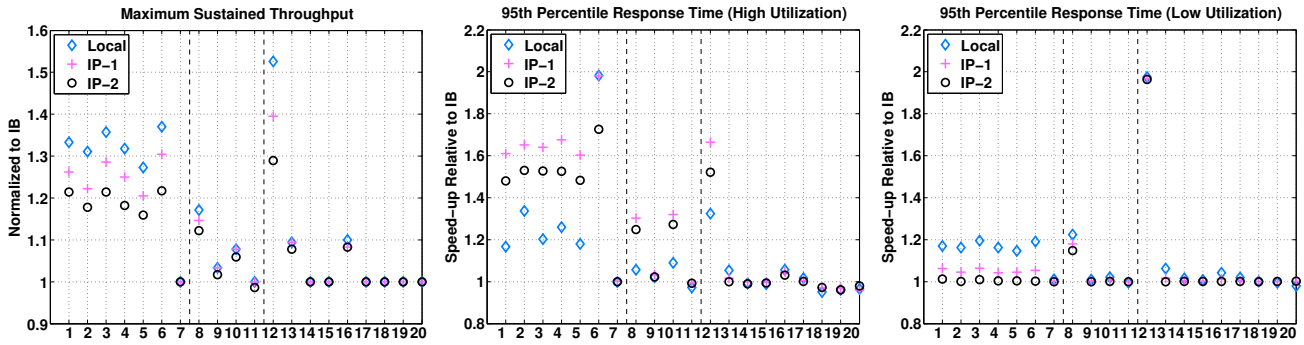

Figure 3: (a) Throughput for varied benchmarks indexed in Table 2. (b, c) 95th percentile response times normalized to IB policy, which permits all NUMA, under high and low server load.

connect bandwidth. The remaining workloads are amenable to a mix of local and remote execution.

5. EVALUATION

We use our models and simulators to evaluate case studies in NUMA mitigation. We find that the optimal refusal policy varies according to a task's memory intensity and system utilization. In addition, we discover that permitting fine-grained NUMA is more energy-efficient than coarse-grained page migration.

Throughput. We analyze throughput by (1) simulating a task and quantifying its performance under three communication scenarios—local, inter-processor, and inter-blade; (2) quantifying the maximum sustainable throughput for the task stream on a blade node. We find that avoiding NUMA reduces task service time and increases throughput.

Figure 3(a) shows how throughput improves as the refusal policy permits more NUMA. A policy that favors local execution will lower service time and increase service. This effect is most pronounced for memory-intensive workloads (S1-S6, M8, P12), which suffer more from NUMA penalties. Ranking policies by increasing throughput gives: **IB**, **IP-2**, **IP-1**, **Local**.

Response Time. Judiciously permitting NUMA reduces queueing delay and response time. Permissive policies, which allow execution when a core becomes available regardless of distance to data, mean less time in arrival and retrial queues. However, latency is not determined by NUMA policy alone. Policy interacts with server load and a task's NUMA sensitivity. We simulate queueing dynamics for highly and lightly loaded servers; system load increases with task arrival rate.

Figure 3 reports the 95th percentile for response time. NUMA execution can be beneficial, even when tasks are memory-intensive (S1-S6, M8, P10) and server load is high. For example, **IP-1** and **IP-2** consistently perform better than **Local** because they improve tasks' likelihood of de-queueing. De-queueing quickly is even more beneficial for compute-intensive tasks (S7, M9, M11, P13-P20), which require little data movement. Refusing NUMA execution does not reduce service time and only increases queueing time. Therefore, compute-intensive tasks can adopt **IB**, which permits load/stores to another blade's memory, with little penalty.

When the server is lightly loaded, many cores are available, queueing time is less important, and service time dominates response time (Figure 3(c)). Refusing NUMA execution only marginally improves response time. **Local** outperforms **IB** and **IP-2** by less than 20% for memory-intensive tasks and by 2%-5% for compute-intensive ones.

Comparison to Data Migration. Task scheduling policies that permit NUMA exploit fine-grained load/store access to remote memory. Remote access retrieves the data needed to fill a cache line instead of migrating entire pages from remote locations into local ones. Although migration is fast, its energy costs are high. We compare the energy costs of remote access and data migration. Our results show that migration energy is 1.5-5.0× greater than that of remote access for memory-intensive workloads.

Energy cost depends on the amount of data transferred and the links used. The processor and queueing simulations report data transferred by remote accesses and tasks that use each link. To estimate the amount of data migrated, we multiply the page size (4KB) by the number of unique pages accessed by a program. Let D_{page} denote the amount of data migrated and D_{access} denote the amount of data supplied to cache lines.

$$E_{\text{M}} = D_{\text{page}}(2E_{\text{mem@100}} + E_{\text{link}}) + D_{\text{access}}E_{\text{mem@20}} \quad (1)$$

Equation 1 estimates page migration energy. Migration accesses DRAM twice, to read from remote memory and to write into local memory. These memory transfers may traverse inter-processor and inter-blade links. The cost of transferring data through DRAM at full bandwidth $E_{\text{mem@100}}$ is 70pJ/bit . The link cost E_{link} ,based on the expected number of inter-processor hops, is 110pJ/bit—see Table 1. Filling cache lines is expensive under low channel utilization and $E_{\text{mem@20}}$ =110pJ/bit [13].

$$E_{\text{R}} = D_{\text{access}}(E_{\text{link}} + E_{\text{mem@20}}) \quad (2)$$

Equation (2) estimates the cost of remote memory accesses. We multiply cache line size (64B) by the number of remote last-level cache misses to estimate the amount of data accessed, D_{access}.

Energy Efficiency. Figure 4 shows that page migration energy is 1.5-5.0× that of remote access, which transfers fine-grained cache lines. However, if migrated data is re-used, we might expect migration to outperform remote execution. The break-even points, where Equations 1 and 2 are equal, are expressed in the number of re-uses.

- Inter-Blade $D_{\text{access}}/D_{\text{page}} = 2.2\times$
- Inter-Processor two-hop $D_{\text{access}}/D_{\text{page}} = 2.9\times$

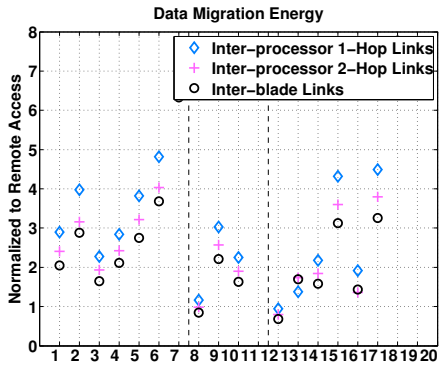

Figure 4: Energy for data migration versus remote access. Benchmarks 18-20 are out of scope; baseline energy is too small for comparison.

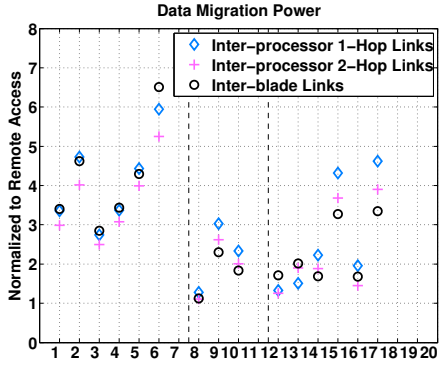

Figure 5: Power for data migration versus remote execution.

- Inter-Processor one-hop $D_{\text{access}}/D_{\text{page}} = 4.8\times$

Figure 5 shows that migration dissipates more power than remote access and is inefficient. Inter-blade accesses increase task delays by 1.6-1.7×, but reduces communication energy by 2.0-8.0× relative to migration. Such energy-delay tradeoffs encourage policies that selectively permit NUMA instead of data migration.

6. CONCLUSION

We analyze communication costs in blade servers, deriving technology models for system simulation. Case studies that coordinate NUMA mitigation and queue management illustrate benefits for throughput, response time, and energy efficiency. Our results motivate further hardware design space exploration and sophisticated task management for in-memory computing on blade servers.

7. ACKNOWLEDGMENTS

This work is partially supported by NSF grants CCF-1149252 (CAREER), CCF-1337215 (XPS-CLCCA), and AF-1408784. This work is also partially supported by STARnet, a Semiconductor Research Corporation Program, sponsored by MARCO and DARPA. Any opinions, findings, or conclusions expressed in this material are those of the author(s) and do not necessarily reflect the views of these sponsors.

8. REFERENCES

[1] AMD. The truth about power consumption starts here. In *AMD White Paper: Power Consumption*, 2009.

[2] J. Antony et al. Exploring thread and memory placement on numa architectures. In *HiPC*, 2006.

[3] B. Goglin and N. Furmento. Enabling high-performance memory migration for multithreaded applications on LINUX. In *IPDPS*, 2009.

[4] T. Ham et al. Disintegrated control for power-efficient and heterogeneous memory systems. In *HPCA*, 2013.

[5] B. Holden. Latency comparison between HyperTransport and PCI-Express in communication systems. In *HyperTransport Consortium Technical Note*, 2006.

[6] R. Hou et al. Cost effective data center servers. In *HPCA*, 2013.

[7] A. Joy et al. Analog-DFE-based 16Gb/s SerDes in 40nm CMOS that operates across 34dB loss channels at Nyquist with a baud rate CDR and 1.2Vpp voltage-mode driver. In *ISSCC*, 2011.

[8] A. Kazmi. Minimizing PCI Express power consumption. In *PCI-SIG Developers Conference*, 2007.

[9] T. Li et al. Efficient operating system scheduling for performance-asymmetric multi-core architectures. In *SC*, 2007.

[10] K. Lim et al. Disaggregated memory for expansion and sharing in blade servers. In *ISCA*, 2009.

[11] K. Lim et al. System-level implications of disaggregated memory. In *HPCA*, 2012.

[12] H. Löf et al. Affinity-on-next-touch: Increasing the performance of an industrial PDE solver on a cc-NUMA system. In *ICS*, 2005.

[13] K. Malladi et al. Towards energy-proportional datacenter memory with mobile DRAM. In *ISCA*, 2012.

[14] D. Meisner et al. BigHouse: A simulation infrastructure for data center systems. In *ISPASS*, 2012.

[15] Micron. Calculating memory system power for DDR3. In *Technicalj Note TN-41-01*, 2007.

[16] PLX Technology. PLX PCIe switch power consumption. In *PLX Technology White Paper*, 2008.

[17] J. Rao, K. Wang, et al. Optimizing virtual machine scheduling in NUMA multicore systems. In *HPCA*, 2013.

[18] J. Regula. Using non-transparent bridging in PCI Express systems. In *PLX Technology White Paper*, 2004.

[19] L. Tang et al. Optimizing Google's warehouse scale computers: The NUMA experience. In *HPCA*, 2013.

[20] M. Zaharia et al. Delay scheduling. In *EuroSys*, 2010.