A HyperTransport-Enabled Global Memory Model For Improved Memory Efficiency

Jeffrey Young and Sudhakar Yalamanchili
School of Electrical and Computer Engineering
Georgia Institute of Technology
jyoung9@gatech.edu, sudha@ece.gatech.edu

Federico Silla and Jose Duato
Universidad Politecnica de Valencia, Spain
{fsilla, jduato}@disca.upv.es

***Abstract*—Modern and emerging data centers are presenting unprecedented demands in terms of cost and energy consumption, far outpacing architectural advances related to economies of scale. Consequently, blade designs exhibit significant cost and power inefficiencies, particularly in the memory system. For example, we observe that modern blades are often overprovisioned to accommodate peak memory demand which rarely occurs concurrently across blades. With memory often accounting for 20% to 40% of the total system power [1], this approach is not sustainable. Concurrently, HyperTransport in concert with new high-bandwidth commodity interconnects can provide low-latency sharing of memory across blades. This paper provides a HyperTransport-enabled solution for seamless, efficient sharing of memory across blades in a data center, leading to significant power and cost savings.**

Specifically, we propose a new global address space model called the Dynamic Partitioned Global Address Space (DPGAS) model that extends previous concepts for Non-Uniform Memory Access (NUMA) and partitioned global address spaces (PGAS). The DPGAS model relies on HyperTransport's low-latency characteristics to enable new techniques for efficient sharing of memory across data center blades. This paper presents the DPGAS model, describes HyperTransport-based hardware support for the model, and assesses this model's power and cost impact on memory intensive applications. Overall, we find that cost savings can range from 4% to 26% with power reductions ranging from 2% to 25% across a variety of fixed application configurations using server consolidation and memory throttling. The HyperTransport implementation enables these savings with an additional node latency cost of 1,690 ns latency per remote 64 byte cache line access across the blade-to-blade interconnect.

I. Introduction

Current trends in today's multi-core processors and data center computing have led to a fundamental shift in how memory can be used and addressed. Improvements in networking technologies have led to a situation where physical network latencies are dropping at a much faster rate than DRAM access times. Additionally, the progression by AMD and Intel to integrate network interfaces closer to main memory and processor cache using HyperTransport (HT) and more recently QuickPath Interconnect (QPI) have dramatically reduced the hardware cost for remote memory accesses. Both of these advances enable the creation of global noncoherent shared memory systems that have previously been commercially available and viable only in high-end supercomputers. This paper introduces a new noncoherent shared memory model called a Dynamic Partitioned Global Address Space (DPGAS), describes the HyperTransport-enabled hardware implementation, and evaluates its implications for cost and power savings in server configurations.

It is anticipated that the number of cores per chip will continue to scale with each technology generation. Coupled with the ubiquitous use of virtualization consolidating applications on-chip, we will see an increase in *memory pressure*, which can be quantified as an increase in memory bandwidth demand and memory footprint. However, technology projections [2] indicate that pin bandwidth and number of I/Os are growing slower than the number of cores on-chip, resulting in reduced available memory bandwidth and physical memory per core.

The current solution to increasing demand for memory footprint on a blade server is to provision memory for the worst case. One recent study empirically measured memory footprints from non-virtualized applications across 3,000 servers under normal applications and found the average physical memory usage to be about 1 Gigabyte of physical memory [3]. However, this study also found that memory requirements can vary greatly from that average amount, with 50% of the studied applications requiring between 1 GB and 4 GB of memory at certain points during the five-week period of data collection. Thus, provisioning server memory for the average case can prove to be inadequate while using worst-case memory provisioning can lead to servers that are substantially over-provisioned.

Furthermore, high-end server DRAM is both expensive and power hungry, especially since the cost of DRAM is a non-linear function of density and memory size. To reduce the cost and power associated with worst-case memory provisioning, memory sharing across blades can be used during periods of peak demand. However, memory sharing via traditional means can exact significant performance penalties through the interconnect and operating system management functions.

The advent of HyperTransport and its integration onto multi-core die and the memory hierarchy reduces the distance from the "wire" to the on-chip memory controller providing low-latency access to remote memory controllers. Memory-to-memory latency is a critical performance determinant of scalable computing systems, and HyperTransport-based solutions have the potential to provide the lowest end-to-end transfer latency for systems composed of tens to thousands of multi-core nodes. However, to productively harness this raw

capability, a global system model must be defined to direct how the system-wide memory is deployed and utilized.

This paper exploits the availability of low-latency memory controllers that are integrated on-chip with HyperTransport to support a global, *noncoherent physical address space* where an application's virtual address space can be dynamically allocated physical memory located on local and remote nodes. Address space management is tightly integrated into the HyperTransport interface to minimize the overhead for remote memory accesses and to permit fast, dynamic changes in address space mappings. Physical memory is dynamically shared by *spilling* memory demand to neighboring blades as necessary during peak periods. Consequently, the total amount of memory to be provisioned can be significantly reduced, leading to substantial cost and power savings with minimal loss of performance (an increase in the page fault rate).

In addition to increased demand for high-performance memory, cores are becoming primitive architectural elements that are no longer the primary determinant of performance. This shift is due to the fact that clock frequency is bound by heat dissipation, and effective instruction issue width is bound by control and data dependencies. Thus, computation scaling will come from the availability of additional cores and thread-level and data-level parallelism. Power dissipation concerns will accelerate the move to simpler streamlined cores, little or no speculation, and doubling of cores across technology generations. Memory bandwidth and interconnection bandwidth will have to track the increase in the number of cores, and thus they will need to be effectively utilized to sustain Moore's Law performance growth with the scaling of cores. Consequently, the DPGAS model is focused on the distribution of memory controllers in the system and their interaction with the interconnection network, which must deliver the lowest latency and highest bandwidth. HyperTransport's high bandwidth and tight integration enable the construction of a system of memory controllers with low-latency access to remote memory.

Specifically, this paper contributes the following:

1) A physical address space model, Dynamic Partitioned Global Address Space (DPGAS), for managing system-wide physical memory in large-scale server systems.
2) Design, implementation, and evaluation of hardware support for the DPGAS model via a memory mapping unit that is integrated with a HyperTransport local interface and tunnels memory requests via commodity interconnect—in this case Ethernet.
3) An brief evaluation of DPGAS with i) traces from memory-intensive applications and ii) an on-demand memory spilling policy to allocate off-blade memory when local demand exceeds available physical memory, and iii) models for cost and power of server DRAM.

II. A Dynamic Partitioned Global Address Space Model

The DPGAS model is a generalization of the partitioned global address space (PGAS) model [4] to permit a flexible, dynamic management of a physical address space at the hardware level—the virtual address space of a process is mapped to physical memory that can span multiple (across blades) memory controllers. The two main components of the DPGAS model are the architecture model and the memory model. The architecture model is memory-centric in the following sense: It is focused on building a network of memory banks that can be accessed with low performance penalties.

A. Architecture Model

Future high-end systems are anticipated to be composed of multi-core processors that access a distributed global 64-bit physical address space. Cores nominally have dedicated L1 caches for instructions and data, but may share additional levels of cache amongst themselves in groups of two cores, four cores, etc. A set of cores on a chip will share one or more memory controllers and low-latency link interfaces integrated onto the die such as HyperTransport [5]. All of the cores also will share access to a memory management function that will examine a physical address and route this request (read or write) to the correct memory controller—either local or remote. For example, in the current-generation Opteron systems, such a memory management function resides in the System Request Interface (SRI), which is integrated on-chip with the Northbridge [6]. Several such multi-core chips can be directly connected via point-to-point links. This is the configuration made feasible by AMD's Opteron series multi-core processors, leading to two-, four-, and eight-socket configurations with low-latency access across two, four, and eight nodes via direct HT connections.

Alternatively, the remote memory controller may not be directly accessible over a few HT links, but rather may be accessible through a switched network such as Infiniband [7] or a custom interconnect such as those employed in high-end computing configurations by Cray [8]. In this case a read or a write operation must be encapsulated into a message and transmitted to be serviced by the remote memory controller that will subsequently generate a response to the local memory controller. In this model, memory controllers receive operations from any core. Effectively, one can view the system as a network of memory controllers.

B. Memory Model

The memory model is that of a 64-bit partitioned global physical address space. Each partition corresponds to a contiguous physical memory region controlled by a single memory controller, where all partitions are assumed to be of the same size. For example, in the Opteron (prior to Barcelona core), partitions are 1 TB corresponding to the 40-bit Opteron physical address. Thus, a system can have 2^{24} partitions with a physical address space of 2^{40} bytes for each partition. Although large local partitions would be desirable for many applications, such as databases, there are non-intuitive tradeoffs between partition size, network diameter, and end-to-end latency that may motivate smaller partitions. Further, smaller partitions may occur due to packaging constraints. For

example, the amount of memory attached to an FPGA or GPU accelerator via a single memory controller is typically far less than 1 TB. Thus, the DPGAS model incorporates a view of the system as a network of memory controllers accessed from cores, accelerators, and I/O devices.

Two classes of memory operations can be generated by a local core: 1) *load/store* operations that are issued by cores to their local partition and are serviced per specified core-semantics, and 2) *get/put* operations that correspond to one-sided read/write operations on memory locations in remote partitions [9]. The get/put operations are native to the hardware in the same sense as load/store operations. The execution of a get operation will trigger a read transaction on a remote partition and the transfer of data to a location in the local partition, while the execution of a put operation will trigger a write of local data to a remote partition. Transactions may have posted or non-posted semantics. The get/put operations are typically visible to and optimized by the compiler. The address space is noncoherent to permit scalability and simplicity. Coherence is separated from the issues central to defining the DPGAS model because large, scalable coherence is still an unsolved research problem, and many systems do not require full-scale coherence across large numbers of servers. Additionally, coherence can be enforced between the one to eight Opteron-based sockets on a server blade to provide local "islands" of coherence. In this case one can view the DPGAS model as dynamically increasing the size of physical memory (across blades) that is associated with a coherence domain although the specific protocols are beyond the scope of this paper.

A sample get transaction on a memory location in a remote partition also requires some knowledge of the underlying network required to transmit the request to and from a remote node. This read transaction must be forwarded over some sort of network to the target memory controller and a read response is transmitted back over the same network. The specific network is not germane to the DPGAS model implementation. However, being constrained by commodity parts, this study utilizes Gigabit Ethernet.

The DPGAS model is very general. For example, once the DPGAS memory model is enabled, an application's (or process's) virtual address space can be allocated a physical address space that may span multiple partitions (memory controllers), i.e., local and remote partitions. The set of physical pages allocated to a process can be static (compile-time) or dynamic (run-time). Multiple physical address spaces can be overlapped to facilitate sharing and communication. This paper is only concerned with a very specific application of DPGAS, namely sharing of memory across blades.

On process creation at a blade or on dynamic memory requests from existing processes, memory can be satisfied by *spilling*—allocating memory from a neighboring blade with spare capacity. We demonstrate in Section V that this simple allocation policy can have a significant impact. The following section addresses the feasibility of a hardware implementation.

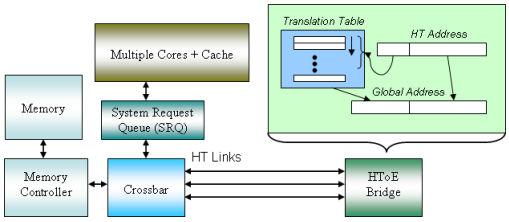

Fig. 1. HToE Bridge with Opteron Memory Subsystem

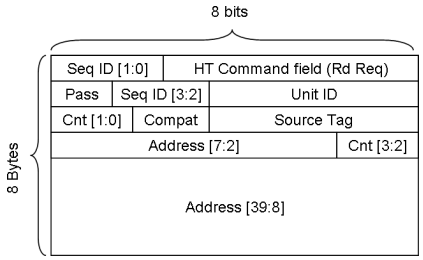

Fig. 2. HT Read Request Packet Format

III. DPGAS: Implementation

Hardware support for DPGAS has two basic components. The first is a memory function that distinguishes between local and remote memory requests. The second is a memory mapping unit that maps remote physical addresses to specific destination memory controllers. The former is available in modern processors such as the Opteron. The latter is contributed by this paper and is tightly integrated into the HyperTransport interface as shown in Figure 1. The proposed memory mapping unit or bridge performs several functions, including 1) managing remote accesses, 2) encapsulating remote requests into an inter-blade communication fabric (the demonstrator uses Ethernet), and 3) extending noncoherent HT packet semantics across nodes. This section describes the design and implementation of the bridge.

A. HyperTransport Overview

HT is a point-to-point packet switched interconnect standard [5] that defines features of message-based communication, including 1) the use of groups of virtual channels, 2) read/write transactions with posted and non-posted semantics, 3) naming and tracking of multiple outstanding transactions from a source, and 4) specification of ordering constraints between messages. In addition and most relevant to our work, the HT specification defines flush and fence commands to manage updates to memory on a node. Our model extends the flush command to a remote version while conforming to normal HT ordering and deadlock avoidance protocols.

A typical command packet is shown in Figure 2, where the fields specify options for the read transaction and preservation of ordering and deadlock freedom. Our implementation specifically relies on the UnitID, SrcTag, SeqID, and address fields. The UnitID specifies the source or destination device and allows the local host bridge to direct requests/responses. The SrcTag and SeqID are used to specify ordering constraints

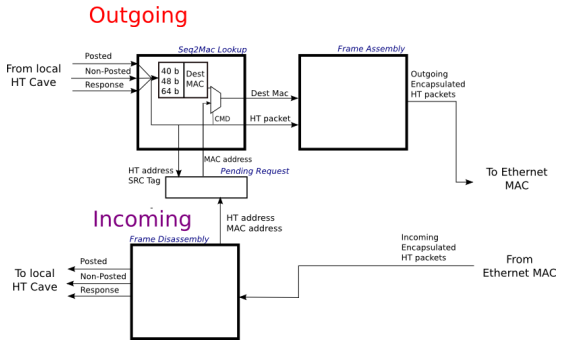

Fig. 3. HToE Bridge Stages

between requests from a device, for example, ordering between outstanding, distinct transactions. Finally, the address field is used to access memory that is mapped to either main memory or HT-connected devices. An extended HT packet can be used that builds on this format to specify 64-bit addresses [5].

B. HyperTransport over Ethernet—Address Translation and Ethernet Encapsulation

Our demonstrator is based on the use of Ethernet as the commodity inter-blade interconnect primarily due to ready availability of hardware implementations. Furthermore, progress on data center Ethernet [10][11] is addressing issues of flow control and error recovery that would enable (with some additional effort) the preservation of HT semantics across blades at low cost. Finally, the bridge design itself does not rely on Ethernet and is easily replaced with other commodity or specialized interconnects. We refer to this demonstrator bridge as the *HT-over-Ethernet (HToE)* implementation. The HToE bridge implementation uses the University of Heidelberg's HyperTransport Verilog implementation [12], which implements an noncoherent HT cave (endpoint) device. Figure 3 shows the stages of the HToE bridge.

The HToE implementation is based on a system with Opteron nodes where each Opteron node has an Ethernet-enabled FPGA card available in the HTX connector slot, such as the University of Heidelberg HTX card [13]. Several nodes are connected via an inexpensive Ethernet switch, and it is assumed that HyperTransport messages sent to remote addresses via the HToE bridge are routed using one of two methods: 1) access to the northbridge address mapping tables (via the BIOS) in order to specify the physical address space mappings for the HToE bridge device, or 2) an intelligent MMU that distinguishes between accesses to the local memory and the I/O address space and HT packets that are sent for non-local addresses through the HToE bridge.

Consider a system that has been properly initialized and consider an application that generates a read operation to an address that is in a remote partition. There are three stages in each individual communication operation (e.g., a read request command) at a given source host and attached devices: 1) extension from the 40-bit physical address in the Opteron to the 64-bit physical address, 2) creation of a HT packet that includes a 64-bit extended address, and 3) mapping the most significant 24 bits in the destination address to a 48-bit MAC address and encapsulation into an Ethernet frame. An efficient implementation could pipeline the stages to minimize latency, but retaining the three stages has the following advantages: 1) It separates the issues due to current processor core addressing limitations from the rest of the system, which will offer a clean, global shared address space, thus allowing implementations with other true 64-bit processors, and 2) it will be easy to port to other platforms that do not encapsulate by using Ethernet frames, but use other link layer formats such as Infiniband. Thus, some efficiency was sacrificed for initial ease of implementation and for a cleaner, modular design.

First, the HT packet type is decoded into a request or response command packet in the module called Seq2Mac in Figure 3. For request packets the two most significant bits of the 40-bit address are decoded to select one of four partition registers to access the 24-bit partition address—the two most significant bits in the 40-bit address used to address the partition register are reset in parallel with the access to the partition register. Now three pieces of information are needed: 1) the extended 24-bit address to form an HT read request packet with extended address, 2) the MAC address of the destination bridge to encapsulate the extended HT packet into Ethernet, and 3) the local MAC address, according to Ethernet frame format to enable the response. Item 3 has been set during initialization, and access to the source MAC address is not in the critical path. Items 1 and 2 have a direct correspondence among them—given a destination node ID or the remote partition address, there is a unique MAC address associated with both data fields. Therefore, the partition register can store both the 24-bit partition address and the destination MAC address together, thus reducing access time when forming the Ethernet frame. Once the remote MAC address and the 64-bit address have been found in the partition table, the new HT packet is constructed and encapsulated in a standard Ethernet packet, illustrated in the figure as the Ethernet Frame Assembly module. The encapsulated packet is then buffered until it can be sent using the local node's Ethernet MAC and the physical Ethernet interface. For packets that send a set amount of data, the control and data packets must be buffered until all the data has been encapsulated into Ethernet frames.

The receive behavior of the bridge on the remote node will require a "response matching" table where it will store, for every non-posted HT request (request that requires a response), all the information required to route the response back to the source when it arrives. This table is required since HT is strictly a local interconnect and response packets have no notion of a destination 40-bit (or extended 64-bit) address. Since the formats of HT request and response packets differ and this implementation desires not to change local HT operation, the SrcTag field of each packet is used to match MAC addresses from an incoming request packet with an outgoing response packet. Note that each request packet contains the source MAC address, and this is the address stored in the

"response matching" table and later used as the destination MAC address for the corresponding response. Encapsulation and buffering occur once again until the response and data can be transmitted over Ethernet. In the HToE bridge, this module is listed as the Pending Request Store in Figure 3 and is shared between incoming and outgoing packets.

It should also be noted that since HT SrcTags are 5 bits, a maximum of 32 outstanding requests can be handled concurrently by this approach. If two request packets arrive with the same SrcTag, then the latter packet is remapped before being stored in the table. When the corresponding response leaves the HToE bridge, the SrcTag is mapped back to its original value to ensure proper HT routing on the requesting local node. Once the response reaches the local HToE bridge that initiated the read request, the HT packet is removed from its Ethernet encapsulation. The UnitID is changed again to that of the local host bridge and the bridge bit is set to send the packet upstream. This allows the local host bridge to route responses to the originating HT device. Other transactions, such as a posted write or a non-posted write, involve similar sequences of events. The differences in these transactions are that for posted writes, no data is stored to create a response; for non-posted writes, only a "TargetDone" response is returned and no data needs to be buffered before the response is sent over Ethernet. Similarly, atomic Read Modify Write commands can be treated as non-posted write commands for the purposes of this model.

IV. DPGAS: Evaluation of Hardware Support

While it may seem simple, memory mapping is on the critical path for remote accesses. This section reports on the evaluation of a hardware implementation of DPGAS support, the bridge, and the integration into the HyperTransport interface and remote extensions to the HyperTransport protocol required to support DPGAS.

A. Bridge Implementation

Xilinx's ISE tool was used to synthesize, map, and place and route the HToE Verilog design for a Virtex 4 FX140 FPGA. Synthesis tests using Xilinx software have indicated that the four modules that make up the bridge are individually capable of speeds in excess of 160 MHz—combined, unoptimized results indicate that the HT bridge is more than capable of feeding a 1 Gbps or faster Ethernet adapter with a 125 MHz (1 Gbps) clock speed. Evaluations for each of the request and reply critical paths suggest that the latency overhead of the bridge is on the order of 24 to 72 ns (for a control packet with no data and a read request response with eight doublewords of data, respectively). In a Xilinx Virtex 4 FX140 FPGA, an unoptimized placement of the bridge uses approximately 1,300 to 1,500 slices, or approximately 5% to 6% of the chip. Overheads that reduced performance included the use of a serial Gigabit Ethernet MAC interface and the use of only one pipeline to handle packets for each of the three available virtual channels. These latency results are listed in Table I along with the associated latency of the Heidelberg cave device from [12]. Total read and write latencies are also listed that incorporate latency statistics discussed in relation to Table II. All bridge operations assume a 125 MHz clock and discount any serialization latency normally associated with Xilinx Ethernet MAC interfaces.

DPGAS operation	Latency (ns)
Heidelberg HT Core (input)	55
Heidelberg HT Core (output)	35
HToE Bridge Read (no data)	24
HToE Bridge Response (8 B data)	32
HToE Bridge Write (8 B data)	32
Total Read (64 B)	1692
Total Write (8 B)	944

TABLE I
Latency Results for HToE Bridge

B. Bridge and Memory Subsystem Latency Performance Penalties

While the HToE bridge proved to be low-latency, it is also important to understand the overall latency penalty that the memory subsystem contributes to remote memory accesses. The latency statistics for the HToE bridge component and related Ethernet and memory subsystem components were obtained from statistics from other studies [6] [12] [14] and from the above place and route timing statistics for our bridge implementation. An overview is presented in Table II. Our HToE implementation was based on a 1 Gbps Ethernet MAC included with the Virtex 4 FPGA, but latency numbers were not available for this IP core. 10 Gbps Ethernet numbers are shown in this table to demonstrate the expected performance with known latency numbers for newer Ethernet standards.

Interconnect	Latency (ns)
AMD Northbridge	40
CPU to on-chip memory	60
Heidelberg HT Cave Device	35 - 55
HToE Bridge	24 - 72
10 Gbps Ethernet MAC	500
10 Gbps Ethernet Switch	200

TABLE II
Latency Numbers Used for Evaluation of Performance Penalties

Using the values from Table I for using the HToE bridge to send a request to remote memory, the performance penalty of remote memory access can be calculated using the formula:

$$t_{rem\_req} = t_{northbridge} + t_{HToE} + t_{MAC} + t_{transmit}$$

where the remote request latency is equal to the time for an AMD northbridge request to DRAM, the DPGAS bridge latency (including the Heidelberg HT interface core latency), and the Ethernet MAC encapsulation and transmission latency.

This general form can be used to determine the latency of a read request that receives a response:

$$t_{rem\_read\_req} = 2*t_{HToE\_req}\ 2*t_{HToE\_resp} + 2*t_{MAC} + 2*t_{transmit} + t_{northbridge} + t_{rem\_mem\_access}$$

These latency penalties compare favorably to other technologies, including the 10 Gbps cut-through latency for a switch, which is currently 200 ns [15]; the fastest MPI latency, which is 1.2 μs [16]; and disk latency, which is on the order of 6 to 13 ms for hard drives such as those in one of the server configurations used below for the evaluation of DPGAS memory sharing [17]. Additionally, this unoptimized version of the HToE bridge is fast enough to feed a 1 Gbps Ethernet MAC without any delay due to encapsulating packets. Likely improvements for a 10 Gbps-comptable version of the HToE bridge would include multiple pipelines to allow processing of packets from different virtual channels and the buffering of packets destined for the same destination in order to reduce the overhead of sending just one HT packet in each Ethernet packet in the current version.

While we assert that the penalties for using DPGAS are low enough to make them attractive for saving memory cost and power, a more detailed study would be required to investigate overall effects on system power due to the fact that an increase in page faults can lead to slower overall execution, costing more static power from other system components. However, there are also other factors that need to be taken into account in this analysis: 1) Page faults are often overlapped with useful computation, so as long as DPGAS does not prohibitively restrict performance, its power and cost savings will not be mitigated by overall system power. 2) One of the basic tenants of using DPGAS for load-balancing type operations is that applications are time-varying, and while some applications may perform slightly worse in the short-term, overall power and cost savings are likely to be higher.

V. DPGAS: Evaluation of Memory Sharing

In the absence of a full hardware testbed, we employ a trace-driven analysis of the potential savings offered by a DPGAS implementation. Virtual address traces were acquired using an instrumented SIMICS model [18] and fed through a page table simulator to determine the number of page faults as a function of physical memory footprints ranging from 32 MB to 1 GB. Five benchmarks were selected: Spec CPU 2006's MCF, MILC, and LBM [19]; the HPCS SSCA graph benchmark [20]; and the DIS Transitive Closure benchmark [21]. These benchmarks had maximum memory footprints ranging from 275 MB to 1600 MB. A 2.1 billion address trace (with 100 million addresses to warm the page table) was sampled from memory intensive program regions of each benchmark. These traces were used in conjunction with application and system models to assess potential power and cost savings.

1) Memory Allocation: Memory allocation is simulated using a simple spill/receive model coupled with random application generation. A random application is picked from the five applications defined above and is added to a server object. Applications are added randomly to servers until a certain failure threshold is reached, i.e., additional applications cannot be added due to lack of free memory on the servers.

In normal allocation, if the server does not have enough free memory, then the allocation would automatically fail. However, in DPGAS allocation, spilling is used to allocate part of a application's memory on a remote neighbor. If one of the nearest remote neighbors cannot allocate enough remote memory to a application, then the allocation fails. The two allocation methods can be compared by running a simulation using normal allocation then using DPGAS with either a) less memory on some of the servers (lower cost/power) or b) a constant amount of memory and more applications in the system (more throughput).

2) Server Models: Two HP Proliant server configurations were selected for analysis, representing high-end and low-end performance points. Both configurations are expected to execute at least 2 instances of VMs per core where a VM instance is modeled as a benchmark application trace. These server configurations are detailed in Table III. All associated system and memory costs and power statistics were derived from [22] and [23].

Model	CPU Cores	Max. Memory	Base Cost/Power
HP DL785 G5	8 quad-core 2.4 GHz Opterons	512 GB	~$42,000/1110 W
HP DL165 G5	2 quad-core 2.1 GHz Opterons	64 GB	~$2,000/197 W

TABLE III
HP Proliant Server Configurations

A. Cost and Power Evaluation

Results from three experiments are shown here, based on results from a memory allocation simulation of random application allocation using both normal and DPGAS memory allocation. Each set of random allocations was performed with normal allocation, and then the same application was allocated using DPGAS. Results for all experiments were averaged over 50 iterations.

1) Fixed Workload and Scale Out: Our first allocation scenario investigated allocating a fixed number of applications onto our high- and low-end server configurations. We based the application numbers on results from Intel's study of candidate applications for virtualization [3], and extrapolated to a data center with 250 servers (which translates to 2,000 or 500 processor sockets for our server configurations) that could support either 19,500 applications using high-end servers or 4,700 applications using low-end servers. Additionally, we investigated the effects of scale out, which is a typical data center method for increasing application capacity. Our experiment started with a scaled-out data center configuration (250 servers) and then consolidated our static number of applications onto 225 servers and 200 servers.

Normal memory allocation started out with enough memory to comfortably support the maximum memory footprint for every application allocated, while DPGAS allocation took into account the amount of unallocated free memory and

decreased the provisioned amount of memory on half of the servers by that amount. This reduction in memory is analogous to designing a data center with half of the servers overprovisioned (receivers in our model) and half of the servers minimally provisioned (spill memory to other nodes). We propose that this non-uniform memory provisioning makes sense in terms of designing data centers, especially since designers already add memory and processors as application memory requirements increase. Using DPGAS helps to reduce initial overprovisioning by allowing for some overprovisioned servers combined with lower cost and power servers, with minimal performance effects due to lack of memory.

The total cost savings for the low- and high-end server configurations are shown in Figures 4 and 5 with savings between normal and DPGAS allocation graphed as the third column of each group. As we see in the base (250-server) case, DPGAS has the potential to save 22% to 26% in memory cost, which translates into a $60,000 savings for the low-end servers and $200,000 for the high-end servers. However, it is also important to notice that savings with DPGAS allocation drop as applications are consolidated onto fewer servers. This is likely due to the fact that there is less free memory unallocated when using normal allocation, and fewer nodes have memory left to act as receiving nodes for remote spilled allocations.

Fig. 4. Scale Out Cost for Proliant DL165 G5

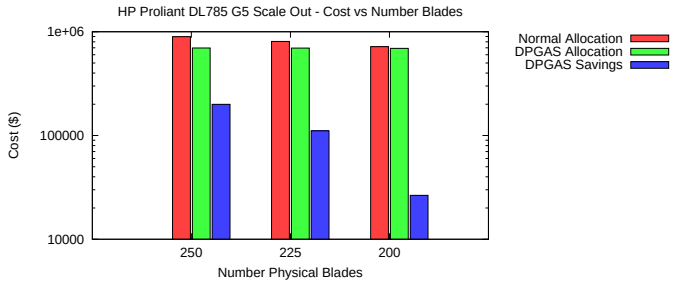

Fig. 5. Scale Out Cost for Proliant DL785 G5

Similarly, the power savings using DPGAS allocation (Figures 6 and 7) is substantial in the base case, with a savings of 3,625 (25%) and 5,875 (22%) watts of input power for the low-end and high-end server configurations, respectively. When server consolidation onto 200 servers is used, this power savings drops substantially to 800 and 500 watts for the same configurations. Both the cost and power results indicate that DPGAS memory allocation is best suited for an environment that has a reasonable amount of fragmented memory that can be utilized.

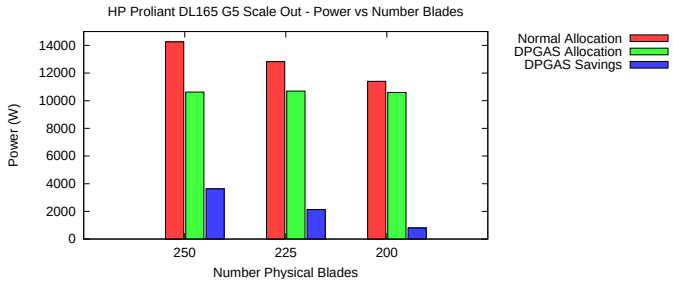

Fig. 6. Scale Out Power for Proliant DL165 G5

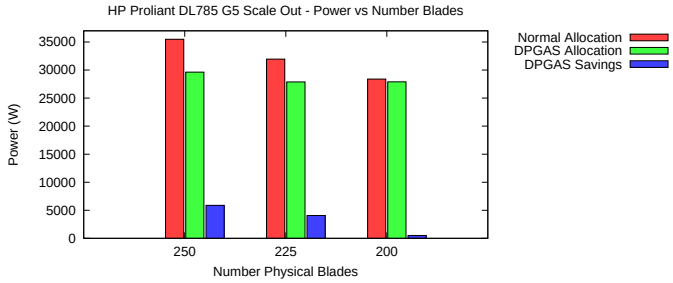

Fig. 7. Scale Out Power for Proliant DL785 G5

2) Memory Throttling: In addition to scaling out data centers, many system designers may also increase the capacity of each server, leading to a scaling up of memory capacity. While this addition of memory can be useful, we also investigate the power, cost, and performance implications of memory throttling—that is, reducing the allocation for each application below its desired level, resulting in additional cost and power savings at the expense of performance, i.e., page faults.

Two additional allocations were tested with fixed applications to compare with our initial fixed application allocation on 250 servers: 1) Each server had 50% of the desired memory and each application was given 50% of its maximum memory footprint, and 2) each server had 25% as much memory, and each application received 25% of its max footprint. The results for cost and power in the high-end server are shown in Figures 8 and 9. The effects of memory throttling are dramatic, and even when less memory is allocated to a server, DPGAS can be used to improve memory cost and power efficiency. For instance, reducing memory from 64 GB to 32 GB in each server reduces memory cost by $478,000 and memory power by 17,750 watts (from a base cost of $897,000 and base power of 35,500 watts). The usage of DPGAS allocation with 50% memory throttling with the high-end server configuration can reduce the total memory cost by $570,000 and total memory power by 21,125 watts.

Additional statistics for the low-end server configuration are shown in Table IV. These experimental results concur with the high-end server configuration, except that power and cost savings are smaller due to less memory fragmentation and less memory overall for remote sharing. Overall, DPGAS

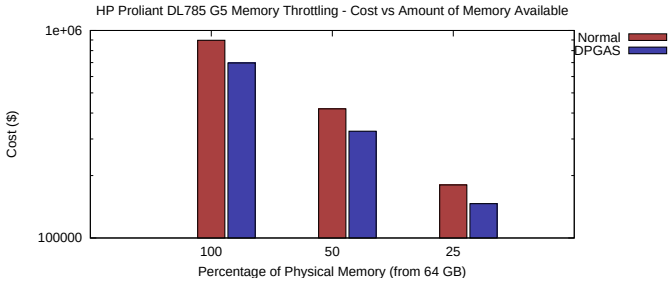

Fig. 8. Scale Up Cost for Proliant DL785 G5

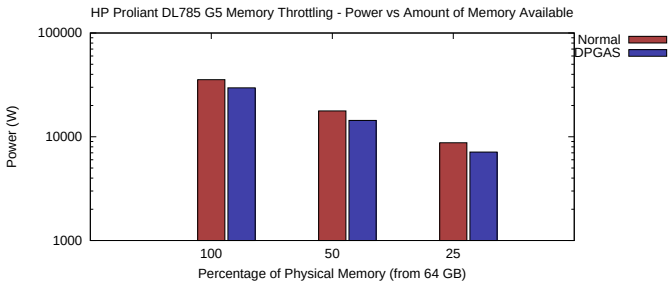

Fig. 9. Scale Up Power for Proliant DL785 G5

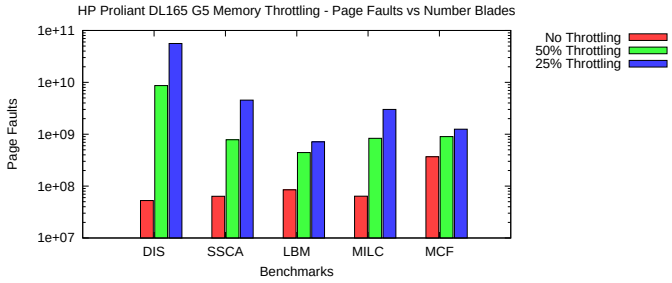

Fig. 10. Memory Throttling Performance for Proliant DL165 G5

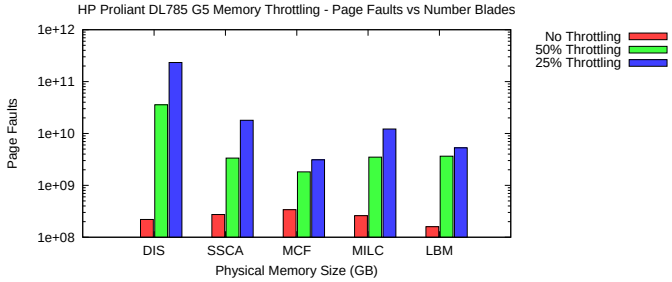

Fig. 11. Memory Throttling Performance for Proliant DL785 G5

enables a 10% to 22% reduction in memory cost and a 16% to 25% reduction in memory power when compared to normal allocation for both the low- and high-end servers.

When using memory throttling, performance must also be taken into account. The results from our trace-driven analysis of the benchmark applications provide data on page fault rates that directly correspond to the amount of memory a benchmark is given. These results are used to generate Figures 10 and 11 that demonstrate the effects of memory throttling on random allocations of each of our benchmark applications. In general, the usage of memory throttling leads to an order-of magnitude increase in the number of page faults for all applications, but some applications with small memory footprints or random access patterns (poor spatial reuse) are affected much more by using memory throttling with normal allocation. However, some initial results from [24] indicate that DPGAS can also be used to improve the performance of applications with throttled memory footprints. If all servers are provisioned with an equal amount of throttled memory (i.e., no cost and power savings), we can use DPGAS to give unallocated memory to applications that would benefit the most from additional memory. By trading some of the power and cost savings, we can reduce the number of page faults system-wide while still garnering the benefits of memory throttling.

Allocation	No Throttling	50% Throttling	25% Throttling
Normal ($)	$230,750	$111,250	$51,500
DPGAS ($)	$171,000	$93,000	$46,250
Normal (W)	14,250	7,000	3,500
DPGAS (W)	10,625	5,875	2,625

TABLE IV
HP PROLIANT 165 G5 COST AND POWER WITH MEMORY THROTTLING

3) Fixed Memory Analysis: In addition to using DPGAS for exploiting the potential for memory cost and power savings, we also investigate the effects of using DPGAS for allocation of a random number of applications on servers with a fixed amount of memory. We tested allocating applications until 250 servers had no available memory for new applications using both normal and DPGAS memory allocation algorithms. Table V shows the results of this experiment. In both the low- and high-end cases, we note that DPGAS allocation of an application failed one to two orders of magnitude fewer times, due to the use of spill/receive DPGAS memory allocation that allows the use of remote memory sharing. Additionally, fixed memory experiments showed that DPGAS can support a small number of additional applications over standard allocation (results not shown in table), typically on the order of 30 to 40 more than normal allocation.

Server Config	Alloc. Method	Workloads Allocated	Alloc. Failures
Proliant 785	Normal	24,875	133
Proliant 785	DPGAS	24,875	3
Proliant 165	Normal	6,163	33
Proliant 165	DPGAS	6,163	2

TABLE V
FIXED MEMORY WORKLOADS

VI. RELATED ISSUES

The discussion of DPGAS thus far has focused more on HyperTransport-level details required to create an efficient global address space, but there are many outstanding software and operating systems research issues that we expect to investigate with this model in future work. Here, we discuss two such issues and possible solutions: the difficulty of implementing a

scalable hardware-based consistency solution and kernel-level support for DPGAS memory allocation.

A. Improved Scalability and Memory Consistency

The DPGAS model focuses on a noncoherent address space implementation in order to ensure high scalability for applications that have limited consistency requirements. However, we would also like to provide support for simple consistency models that can assist application programmers in ensuring program correctness.

Strict memory consistency, such as that which is offered by sequential consistency models, is difficult to support in a scalable manner, especially when using encapsulated point-to-point technologies like HyperTransport. HyperTransport does not require strict ordering of packets on HT links, and the ordering requirements for virtual channels mean that request and response packets can possibly arrive out of order at remote destinations. Additionally, an Ethernet implementation complicates consistency support due to its current lack of flow control and ordered delivery.

While the use of new data center Ethernet standards or other interconnects such as Infiniband can be used to correct ordering problems over the encapsulation network, extensions to the initial bridge implementation may be required to help order HyperTransport packets and to push them to their remote destination. Anticipating future needs for building these models, we have incorporated support for standard HyperTransport atomic operations and a modified HyperTransport operation called *remote flush* in our bridge implementation. Remote flush works by receiving a standard HyperTransport posted request with a special address bit combination on the outgoing path of the bridge and then generating a HyperTransport flush request to send to a remote node. Once the remote flush packet is de-encapsulated on the remote node, it proceeds to push posted requests with the same SrcTag upstream.

In addition to this hardware support, we also advocate that relaxed consistency models be investigated for use with HT-based DPGAS, rather than solely focusing on sequential consistency. This design choice along with limited support for coherence will allow for DPGAS clusters to be much more scalable.

B. Operating System Support and Memory Allocation with DPGAS

In addition to memory consistency support, there are many research issues associated with both memory allocation and operating system support. Our initial assumption is that the memory function in DPGAS will use messages to communicate infrequent memory allocation requests and configuration changes to the address translation portion of the HToE bridge. However, there is much room for improvement in terms of using application profiles and memory footprint statistics to make better choices about memory allocation.

For instance, applications running in paravirtualized VMMs like Xen typically trap to the hypervisor when page faults occur. Although the mechanism for trapping page faults and swapping pages differs from standard Linux, profiling mechanisms could be inserted into Xen to track the frequency of virtual pages being swapped in and out. This profile could then be used to devise a memory allocation algorithm that maps frequently swapped virtual addresses to local physical pages and infrequently used addresses to remote pages. Additionally, gradual page migration could be used to move virtual address mappings back to local memory as application patterns change, taking full advantage of the time-varying nature of enterprise applications. This profiling mechanism can be combined with kernel-level memory functions, such as that implemented in [25] to provide a simple but effective way of using DPGAS at the operating systems level.

Memory allocation is another issue that is being actively researched in relation to DPGAS. In addition to the spill/receive model discussed earlier, timed memory leases could be used to fairly share memory between many different requesting nodes. We envision a large number of different memory allocation algorithms could be used with DPGAS with each one focused on optimizing a different metric, such as cost, power, performance, or even failure recovery (via techniques like replication).

VII. Related Work

Other researchers have also been focused on the growing power and cost implications of large clusters and server farms. Feng, et al [26] discussed the efficiencies associated with large servers and proposed a power-efficient supercomputer called Green Destiny. Other strategies have included dynamic voltage scaling for power-aware computing [27] with a focus on CPU power. Raganathy, et al [28] have also suggested that power-management should take place at the server enclosure levels so that individual systems are not overprovisioned. This study also focused mainly on high-level CPU power management, not memory power.

However, Lefurgy's 2003 study [1] cited important reasoning behind why DRAM cost and power should be considered as a major component in improving overall server efficiencies. Several other researchers have also begun focusing on memory power management at the architecture level, including [29], which proposes using adaptive power-based scheduling in the memory controller, and [30], which uses power "shifting" driven by a global power manager to reduce power of the overall system based on runtime applications.

At the operating system level, [31] proposed a power-aware paging method that utilizes fast MRAM to provide power and performance benefits. Tolentino [32] also suggested a software-driven mechanism to limit application working sets at the operating system level and reduce the need for DRAM overprovisioning.

An evaluation of power and cost trends similar to the ones in this paper was conducted in [33], concluding that separate PCI Express-based memory blades could be used to reduce overall memory usage and memory cost and power. [34] investigated real-world statistics for some of the large "warehouse-sized" server farms that Google runs. However, no current studies

have focused on the potential for cost and power savings based on low-latency integrated networks.

VIII. Conclusion

With increasing server power and cost outpacing related performance gains, a focus on making data centers and clusters as efficient as possible is vital from a business perspective. We present a new address space model, a Dynamic Partitioned Global Address Space, that defines a dynamic hardware-based address translation scheme for efficiently utilizing remote memory with low-latency interconnects such as HyperTransport. An implementation of this model has been demonstrated by encapsulating HyperTransport packets in Gigabit Ethernet via our HT over Ethernet bridge, and initial synthesis results indicate that remote read and write operations are low-latency and comparable to the fast message-passing implementations.

Additionally, we demonstrate the effectiveness of DPGAS to save system cost and power by allowing for the use of fewer memory DIMMs. DPGAS using a spill/receive allocation can be combined with other memory-saving techniques such as server consolidation or memory throttling to produce additional benefits over a normal memory allocation scheme, and DPGAS allocation also has potential to improve allocation for a fixed memory scenario. Finally, we discuss future research issues for operating system-level memory allocation schemes and our bridge's hardware support for building basic memory consistency models. In addition to building better software support for DPGAS, we would also like to investigate hardware implementations of the HT over Ethernet bridge using Infiniband and the University of Heidelberg's Extoll network layer and optical link-based HTX cards.

References

[1] C. Lefurgy, K. Rajamani, F. Rawson, W. Felter, M. Kistler, and T. W. Keller, "Energy management for commercial servers," *Computer*, vol. 36, no. 12, pp. 39–48, 2003.

[2] "International technology roadmap for semiconductors, executive summary 2007." [Online]. Available: http://www.itrs.net

[3] S. Chalal and T. Glasgow, "Memory sizing for server virtualization," 2007.

[4] P. Charles, C. Grothoff, V. Saraswat, C. Donawa, A. Kielstra, K. Ebcioglu, C. von Praun, and V. Sarkar, "X10: an object-oriented approach to non-uniform cluster computing," in *OOPSLA '05: Proceedings of the 20th annual ACM SIGPLAN conference on Object oriented programming, systems, languages, and applications*. New York, NY, USA: ACM, 2005, pp. 519–538.

[5] "Hypertransport specification, 3.00c," 2007. [Online]. Available: http://www.hypertransport.org

[6] P. Conway and B. Hughes, "The amd opteron northbridge architecture," *IEEE Micro*, vol. 27, no. 2, pp. 10–21, 2007.

[7] I. T. Association, "Infiniband architecture specification volume 1. release 1.2.1," 2008. [Online]. Available: http://www.infinibandta.org

[8] (2008) Cray xt3 supercomputer scalable by design. [Online]. Available: "http://www.craysupercomputers.com"

[9] J. Nieplocha, R. J. Harrison, and R. J. Littlefield, "Global arrays: a portable "shared-memory" programming model for distributed memory computers," in *Supercomputing '94: Proceedings of the 1994 ACM/IEEE conference on Supercomputing*. New York, NY, USA: ACM, 1994, pp. 340–349.

[10] "Ieee standard for local and metropolitan area networks: Virtual bridged local area networks," 2006. [Online]. Available: http://standards.ieee.org

[11] "Ieee working group - 802.1qau: Congestion notification," 2008. [Online]. Available: http://www.ieee802.org

[12] D. Slogsnat, A. Giese, M. Nüssle, and U. Brüning, "An open-source hypertransport core," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 1, no. 3, pp. 1–21, 2008.

[13] U. Bruening, "The htx board: the universal htx test platform." [Online]. Available: http://www.hypertransport.org/members/u_of_man/htx_board_data_sheet_UoH.pdf

[14] "Intel 82541er gigabit ethernet controller." [Online]. Available: http://download.intel.com

[15] "Quadrics qs ten g for hpc interconnect product family," 2008. [Online]. Available: http://www.quadrics.com/Quadrics/QuadricsHome.nsf/NewsByDate/787FABD3533E276580257228004933F4

[16] "Mellanox connectx ib specification sheet," 2008. [Online]. Available: http://www.mellanox.com

[17] "Storagereview.com drive performance resource center," 2008. [Online]. Available: http://www.storagereview.com/

[18] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hållberg, J. Högberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A full system simulation platform," *Computer*, vol. 35, no. 2, pp. 50–58, 2002.

[19] J. L. Henning, "Spec cpu2006 benchmark descriptions," *SIGARCH Comput. Archit. News*, vol. 34, no. 4, pp. 1–17, 2006.

[20] D. A. Bader and K. Madduri, "Design and implementation of the hpcs graph analysis benchmark on symmetric multiprocessors," in *HiPC*, 2005, pp. 465–476.

[21] "Dis stressmark suite, updated by uc irvine," 2001. [Online]. Available: "http://www.ics.uci.edu/~amrm/hdu/DIS_Stressmark/DIS_stressmark.html"

[22] "Hp power calculator utility: a tool for estimating power requirements for hp proliant rack-mounted systems," 2008. [Online]. Available: http://h20000.www2.hp.com/bc/docs/support/SupportManual/c00881066/c00881066.pdf

[23] "Hp proliant dl servers - cost specifications," 2008. [Online]. Available: http://h18004.www1.hp.com/products/servers/platforms/

[24] J. Young, "Dynamic partitioned global address spaces for high-efficiency computing," 2008. [Online]. Available: http://etd.gatech.edu

[25] S. Liang, R. Noronha, and D. K. Panda, "Swapping to remote memory over infiniband: An approach using a high performance network block device." September 2005.

[26] W. chun Feng, "Making a case for efficient supercomputing," *Queue*, vol. 1, no. 7, pp. 54–64, 2003.

[27] R. Ge, X. Feng, and K. W. Cameron, "Improvement of power-performance efficiency for high-end computing," in *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 11*. Washington, DC, USA: IEEE Computer Society, 2005, p. 233.2.

[28] P. Ranganathan, P. Leech, D. Irwin, and J. Chase, "Ensemble-level power management for dense blade servers," in *ISCA '06: Proceedings of the 33rd annual international symposium on Computer Architecture*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 66–77.

[29] I. Hur and C. Lin, "A comprehensive approach to dram power management," in *HPCA '08: Proceedings of the 14th annual International Symposium on High-Performance Computer Architecture*, 2008.

[30] W. Felter, K. Rajamani, T. Keller, and C. Rusu, "A performance-conserving approach for reducing peak power consumption in server systems," in *ICS '05: Proceedings of the 19th annual international conference on Supercomputing*. New York, NY, USA: ACM, 2005, pp. 293–302.

[31] Y. Hosogaya, T. Endo, and S. Matsuoka, "Performance evaluation of parallel applications on next generation memory architecture with power-aware paging method," *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pp. 1–8, April 2008.

[32] M. E. Tolentino, J. Turner, and K. W. Cameron, "Memory-miser: a performance-constrained runtime system for power-scalable clusters," in *CF '07: Proceedings of the 4th international conference on Computing frontiers*. New York, NY, USA: ACM, 2007, pp. 237–246.

[33] K. Lim, P. Ranganathan, J. Chang, C. Patel, T. Mudge, and S. Reinhardt, "Understanding and designing new server architectures for emerging warehouse-computing environments," in *ISCA '08: Proceedings of the 35th International Symposium on Computer Architecture*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 315–326.

[34] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," in *"ISCA 2007: Proceedings of the 34th annual international symposium on Computer architecture"*. New York, NY, USA: ACM, 2007, pp. 13–23.