Dynamic Partitioned Global Address Spaces for Power Efficient DRAM Virtualization

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

***Abstract*—Dynamic Partitioned Global Address Spaces (DPGAS) is an abstraction that allows for quick and efficient remapping of physical memory addresses within a global address space, enabling more efficient sharing of remote DRAM. While past work has proposed several uses for DPGAS [1], the most pressing issue in today's data centers is reducing power. This work uses a detailed simulation infrastructure to study the effects of using DPGAS to reduce overall data center power through low-latency accesses to "virtual" DIMMs. Virtual DIMMs are remote DIMMs that can be mapped into a local node's address space using existing operating system abstractions and low-level hardware support to abstract the DIMM's location from the application using it. By using a simple spill-receive memory allocation model, we show that DPGAS can reduce memory power from 18% to 49% with a hardware latency of 1 to 2 μs in typical usage scenarios. Additionally, we demonstrate the range of scenarios where DPGAS can be realized over a shared 10 Gbps Ethernet link with normal network traffic.**

***Keywords*-memory allocation; DRAM power; interconnects; power efficiency**

I. Introduction

The emergence of cloud computing and the continued growth of data centers have been accompanied by increased power requirements. While many architectural improvements in data center servers have focused on reducing power consumed by CPU and disk, several studies [2] [3] indicate that DRAM power can consume up to 30% of the total hardware power budget and thus should not be overlooked. One of the challenges in optimizing memory power usage is the time-varying nature of workloads. Servers within the data center are usually overprovisioned to handle memory footprints associated with peak workloads leading to excess static power dissipation. Furthermore, commodity server blades are packaged in a manner that remote blades can share only via heavyweight, OS-coordinated mechanisms between blades - Remote Direct Memory Access (RDMA) and custom NUMA interconnects have provided two approaches to make DRAM sharing easier, but they are limited by registration overhead and the requirement for non-commodity parts, respectively.

We previously proposed a model for non-coherent global address spaces called Dynamic Partitioned Global Address Spaces (DPGAS). Virtual memory address spaces can be allocated across memory controllers on distinct blades and can be transparently supported by low level communication mechanisms over commodity networks. Our previous work focused on analytical models to show the feasibility of using DPGAS to reduce memory power and improve cost efficiency in large data centers. This work contributes a more detailed analysis through the trace-driven simulation of several workloads and memory allocation across blades. We study the effects of the DPGAS model on memory performance, network latency, and utilization, specifically in environments with existing network loads.

In addition to having implications for reducing DRAM power, DPGAS also enables memory virtualization at the hardware level. DIMMs can be virtualized transparently to applications using interactions between an OS process, such as a kernel-level block device or a hypervisor module, and the hardware required to set up and release remote memory partitions. Other existing memory virtualization solutions [4] have shown the benefits of using memory virtualization as a technique for data caching of extremely large datasets such as in-memory databases.

The rest of this paper addresses the following themes: First, we review the trends that enable HyperTransport over Ethernet (HToE) and the definition of how DPGAS works. Next, we look at the features of our hardware model and address the creation of a new simulation infrastructure to study DPGAS in more detail. Finally we investigate the effects of using DPGAS on reducing memory power and also on memory and network utilization.

II. Important Trends Enabling HToE and DPGAS

HyperTransport over Ethernet is part of an important shift toward integrating networks and memories that has resulted from several trends. These trends include the dramatic decreases in network end-to-end latencies, the integration of high-speed on-die networks, and the pressing need for additional pin bandwidth in future architecture designs.

In the past two decades, raw network packet latencies have dropped from milliseconds to hundreds of nanoseconds while DRAM access latencies have remained relatively constant (due to increases in density and a focus on increasing DRAM bandwidth). Additionally, high-speed network interfaces such as Intel's QuickPath Interconnect [5] and AMD's HyperTransport [6] are being integrated on-die, further reducing congestion-free memory-to-memory hardware latency between remote memory modules. The integration of high-performance network interfaces into the die and the consequent minimization of hardware memory-to-memory latency between blades makes the sharing of physical memory across blades practical.

In contrast to this trend of continually increasing on- and off-die interconnect bandwidth and reduced latency, the trend of increased VM consolidation and multi-core proliferation has led to reduced pin bandwidth per core and per hardware thread. As the number of VMs per core and the core count increases, the demand for memory bandwidth will also increase for on-die memory controllers. Integrated, very low-latency interconnects will have to serve as a pressure valve to release memory pressure by providing low-latency access to remote memory.

HyperTransport over Ethernet provides the benefits of a low-latency interconnect with the additional benefits of providing this "release valve" for additional memory bandwidth. HyperTransport was chosen due to its open specification, while Ethernet was selected for its wide deployment, low cost, and its growing usage as an encapsulation and convergence standard for other traditional data center technologies such as FibreChannel over Ethernet (FCoE) [7] and RDMA over Converged Ethernet (RoCE).

III. Dynamic Partitioned Global Address Spaces

Although the concept of Dynamic Partitioned Global Address Spaces (DPGAS) is covered in greater detail in [1], we review the basic concepts here.

The DPGAS model is a generalization of the partitioned global address space (PGAS) model [8] to permit 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.

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. 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 or QuickPath. All of the cores will also 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 Queue (SRQ), which is integrated on-chip with the Northbridge [9].

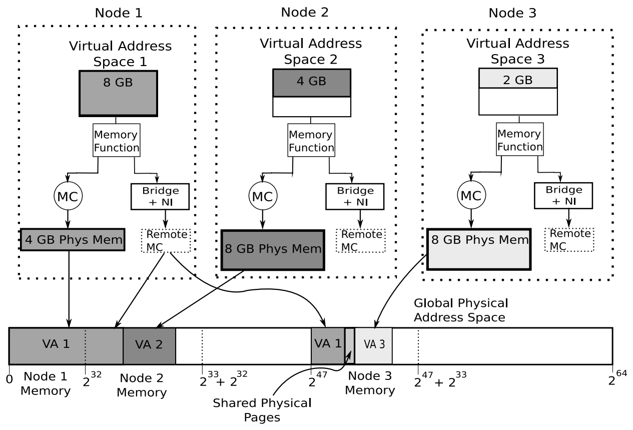

Figure 1. DPGAS Overview and Memory Functions

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, 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. Thus, the DPGAS model incorporates a view of the system as a network of memory controllers accessed from cores, accelerators, and I/O devices.

The DPGAS model operates over a non-coherent global physical address space. However, if blades implement coherence, one can view the DPGAS model as dynamically increasing the size of physical memory accessible by a blade, i.e., within a coherence domain. Specific protocols are beyond the scope of this paper.

The programming model provides *get/put* operations that correspond to one-sided read/write operations on memory locations in remote partitions [8]. A sample *get* transaction on a memory location in a remote partition must be forwarded over some sort of network to the target memory controller and a read response is transmitted back over the same network. 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.

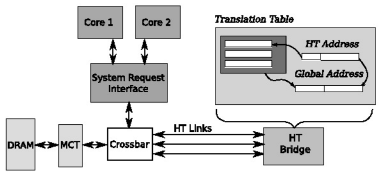

Figure 2. DPGAS Hardware Support in Relation to Northbridge

A visual representation of this overlapping of global addresses to share pages is shown in Figure 1.

IV. DPGAS: Implementation Using HyperTransport over Ethernet

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 through features like the Opteron's System Request Queue (also sometimes called the System Request Interface). A generic overview of the locations of these components can be seen in Figure 2.

A. *HToE Hardware Overview*

As part of our earlier effort, we created a prototype Verilog module to demonstrate the basic performance impact of using HToE and DPGAS for transparent memory sharing. The prototype of our HyperTransport to Ethernet Adapter (HTEA) demonstrated support for 1 Gbps Ethernet with a single pipeline that operated in excess of 125 MHz. The original implementation of the HTEA was designed to fit into an FPGA chip that was part of the University of Heidelberg's HTX card [10], which combines their custom HT cave device in a card that also has an FPGA and an Ethernet NIC. The HTEA handles the encapsulation and routing of HToE packets, and the HTX card provides the necessary hardware to interface HT and Ethernet. The processing pipeline in the HT interface adds approximately 8-12 cycles for processing packets - address mapping and transaction management. The direct coupling to HT provides very low-latency injection and ejection into commodity networks. A more detailed description can be found in [1].

B. *DPGAS Simulation Infrastructure*

Figure 3 shows an illustration of the overall simulation environment, which takes advantage of the open-source NS-3 network simulator project [11] and University of Maryland's DRAMSIM [12]. Scalable simulation is supported by the event-driven scheduling classes used within NS-3 as well as current plans to release a distributed NS-3 simulator kernel. Detailed models in DRAMSIM and NS-3 allow us to model power and latency used by DDR2 DRAM as well as per-hop latencies associated with different topologies of switched Ethernet.

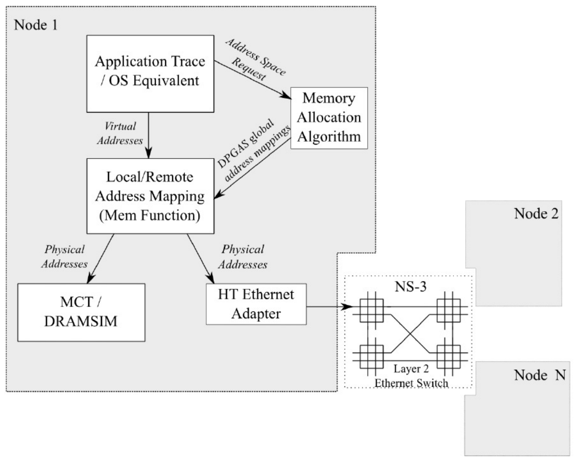

Figure 3. DPGAS Infrastructure

In addition, our custom code provides a detailed model of the HTEA and the Opteron Northbridge's System Request Queue (SRQ) that can be used to map physical addresses to local or remote DRAM [13] [9]. The SRQ is programmed to determine whether a physical address is mapped to the local DRAM and memory controller or to another device attached to the Northbridge's crossbar. By using the address mapping tables, we can specify that a certain subset of physical addresses map to the HTEA and are satisfied by remote memory controllers. Our infrastructure uses a C++ mapping function that simulates the operation and address range updates in the Opteron (SRQ).

C. *Memory Allocation Algorithms*

Memory allocation algorithms for DPGAS are envisioned to be implemented at the operating system level using a simple kernel device that communicates via MPI or by standard Unix sockets. The OS is notified of changes to its available physical memory using the capabilities of libraries like Linux's libnuma or, in the case of a virtual OS, updates to the hypervisor's page tables. The memory allocation daemon negotiates for changes to the global physical address space with daemons on other nodes and then provide dynamic mapping updates to the System Request Queue using Function 1 Address Map Registers [13] and posted HT packets to the HTEA. Updates to the underlying hardware and operating system allow for transparent remapping of remote memory accesses without requiring application involvement. Since our current infrastructure uses synthetic application traces instead of an actual operating system, we have labeled this daemon as the memory allocation algorithm in Figure 3.

We have implemented a spill-receive model for memory allocation wherein the local node spills memory requests to neighboring nodes. The envisioned process for memory allocation with DPGAS is briefly summarized here: 1) On system boot, DPGAS daemons (OS-level or hypervisor-level

Table I
TEST OVERVIEW

Num Nodes	Apps / node	Spill Nodes	Receive Nodes	Mem Size (GB)	Apps / Receive Node
1	4,16	0	0	4,16	N/A
2	6	0	0	4/8	2
2	6,16	1	1	4,16	2,8
4	6,16	2	2	4,16	2,8
8	6,16	4	4	4,16	2,8
16	16	0	0	16/20	8
16	6,16	8	8	4,16	2,8
8	6,16	6	2	4 (8 on Recv Nodes),16	2,8
16	6,16	14	2	4 (8 on Recv Nodes),16	2,8

process) are started on each DPGAS-enabled machine, and each blade enumerates its closest (one-hop) neighbors. 2) The DPGAS daemon monitors current DRAM allocation and load, keeping track of average load or major page faults (to disk) over a long period of time (similar to running "ps" in the background of Linux OS). 3) When the average allocation exceeds a preset threshhold (e.g. 80% of DRAM or 100 page faults to disk in 4 seconds), the DPGAS daemon issues a request to a neighboring node (or a dedicated "receive" node) via message passing for M MB of memory. 4) The daemon at neighboring Node B checks to see if it can satisfy the DPGAS memory request and then decides how much of the requested M MB of memory can be allocated. 5) Node A is notified whether its spill request is rejected or fulfilled and both nodes update their HTEA and SRQ mapping tables and notify the local VMM or OS of the change in the mapped physical address space.

D. Memory Deallocation

Although our current simulation infrastructure does not enforce tie-breaking for simultaneous DPGAS requests, we also must provide for deallocating memory that has been mapped across virtualized DIMMs using DPGAS. Memory deallocation takes place in three situations: 1) the requesting node has finished using the remote memory it requested. Typically, this takes place when its load has decreased below a certain threshold or a hard timeout (set globally for all nodes at start up) occurs. 2) The node that is receiving, i.e. sharing memory via DPGAS, experiences high load (typical when all nodes have equal amounts of memory). 3) Another Node C with better fairness characteristics (fewer requests or a smaller amount of memory requested) needs memory that is currently being used by the original requesting Node A. Obviously, the manner in which memory is forcefully deallocated depends on the overall implementation goals and required QoS metrics of the system, but one possible deallocation technique is as follows: 1) Node B notifies node A that it is revoking its shared memory. Node A must then ACK back to confirm it is done. If it does not ACK back within a preset time limit, Node B will change its HTEA mapping tables ensuring that incoming requests to Node B are rejected. 2) Both nodes update their HTEA mappings, VMM status, and SRQ tables. 3) Node B proceeds to allocate memory to the next highest-priority node.

V. TEST SETUP

Synthetic traces were generated based on application access patterns and inter-reference timing from other studies looking at benchmark suites such as Spec 2006 [14]. These studies were used to specify the DRAM access inter-reference time for synthetic benchmarks. The Spec 2006 benchmark represents applications that are worst-case application workloads in terms of L2 cache misses, and some of the most intensive of these applications, such as MCF and Omnetpp, have an L2 cache miss every 4000-4500 cycles for a 2 MB L2 cache size. Enterprise applications and general-purpose applications likely have better spatial and temporal locality than the SPEC suite, so we use an inter-reference time ranging from 10,000 to 20,000 cycles between L2 cache misses (memory accesses).

Three different synthetic benchmarks were used to represent different access patterns: 1) Random - worst case scenario where there is no spatial locality within an address stream; 2) Clustered Random - accesses are clustered around a mean address, and locality is similar to an application repeatedly accessing several pages in the application's address space; 3) Strided - the access stream is specified by a stride and is related to applications that perform large numbers of sequential array element accesses. Application footprints were randomly assigned from 500-1500 MB, although specific tests were designed with larger memory footprints to have one or more applications that would require spilling to a remote node via DPGAS. The sizing of these memory footprints was drawn from insights found in [15]. The number of read and write operations was split equally between reads and writes.

DRAMSIM was initialized using a 4 GB module that corresponds to Micron's MT47H512M8 TwinDie 4 GB DDR2 module [16]. Timing and power parameters were selected for the DDR2-800 part, and 1 and 4 ranks were used to simulate 4 and 16 GB of installed memory, respectively. The associated CPU speed was set at 3000 MHz.

NS3 was initialized with between one and 16 nodes, with each node connected to a single switch via 10 Gbps channels. NS3's raw socket layer was used to represent the transfer of HT packets inside Ethernet packets. Simulations were run for 100 real-world seconds and each node had between one and 16 applications that each generated 500,000 synthetic addresses. For applications with a miss rate of every 20,000 cycles, this would mean that the synthetic applications would run about 3 real-world seconds. Tests with 4 GB were run with 6 applications per "spill" node and 2 applications per "receive" node, while tests with 16 GB of memory were run with at least 16 applications for the "spilling" node and 8 applications for the "receiving" node. In each test a certain number of applications were specified to spill to other nodes, usually one to three applications that exceeded the amount of memory for a minimally provisioned blade. This setup assumes that an overprovisioned blade would have 8 GB and 20 GB of memory, while our simulation uses 4 GB and 16 GB, respectively.

Two tests were also devised to test scenarios similar to those tested by proponents of memory blades [17] for times when a greater number of "client" servers would like to access remote memory on one central blade to support peak workloads. The first simulation used eight nodes, with six nodes containing 4GB of installed memory spilling to two nodes with 8 GB of DRAM installed. The second test used sixteen nodes with fourteen spill nodes and 2 receive nodes with 4 and 8 GB of installed DRAM, respectively.

The simulations tracked several different metrics including: 1) dynamic power for DRAM (W), 2) memory access latency for DRAM (ns), 3) link and buffering latency for the 10 Gbps network (ns), and 4) network utilization of the 10 Gbps network (B/s). These results are discussed more in Section VI.

The different test cases are summarized in Table I.

VI. Results

We break down the results according to the different metrics our simulations measured for memory and the network.

A. Memory Latency

Table II shows the average memory access latency across all simulations. The use of DPGAS spilling does not dramatically change the average latency for a memory access, although it can cause a slight increase in average latency of a lightly loaded node that receives DPGAS remote memory request. Changes in access patterns from DPGAS spill operations and increases in the number of applications also affect the number of conflicts on open rows in the simulated DRAM, requiring additional latency to access a new row. In most simulations, access latencies varied between 60 and 70 nanoseconds. Variations between nodes are expressed in the standard deviation calculations in Table II.

Table II
Memory Latency

Simulation, DRAM Size	Ave. Mem Latency (ns)	Std. Dev.
2 node, 4/8GB	54.28	6.21
2 node, 4GB	53.06	2.5
4 node, 4GB	69.42	5.58
8 node, 4GB	66.29	9.5
16 node, 4GB	64.35	11.5
8 node, 4GB,8GB	67.74	10.89
16 node, 4GB,8GB	69.98	10.65
2 node, 16GB	68.11	12.51
4 node, 16GB	68.27	13.28
8 node, 16GB	68.72	14.21
16 node, 16/20GB	68.17	9.11
16 node, 16GB	68.84	7.2

Further tests showed that the latency difference for DPGAS versus non-DPGAS simulations (two nodes with 4/8 GB and sixteen nodes with 16/20 GB) led to an increase (sixteen node case) or decrease (two node case) of less than 2 ns and a smaller standard deviation. While the amount of traffic to the "receiving" DRAM increased, this DRAM was already underutilized due to lighter application loads. The decrease in standard deviation reflects that the average latency of memory accesses on the "spill nodes" was reduced as remote accesses took place.

B. Memory Power

DRAMSIM tracks the average power for one activation-to-activation cycle, meaning that it tracks the power used to perform one operation, whether it be a read, write, precharge, or refresh operation. The calculated power is averaged across all the accesses that take place in the memory simulation to get a power value for each type of operation the DRAM performs. DPGAS-enabled servers can be used to reduce the static component of DRAM power, that is the average power that is required even if a DIMM is not used for read or write operations. Due to overprovisioning, DIMMs that are underutilized must still be refreshed, and they still consume leakage energy.

This static component of DRAM power includes the power used when a DIMM is in power-down or standby states for either active (when data is stored in the sense amplifiers) or precharge (all data is restored to a row) modes. Additionally, static power takes into account the refresh power for refresh cycles that occur at regular intervals every 64 ms (for our model DIMM). Figures 4 and 5 show the average static power savings achieved by reducing the DRAM in each node by 1 DIMM. For the 4 GB case, we assume that power savings are calculated based on

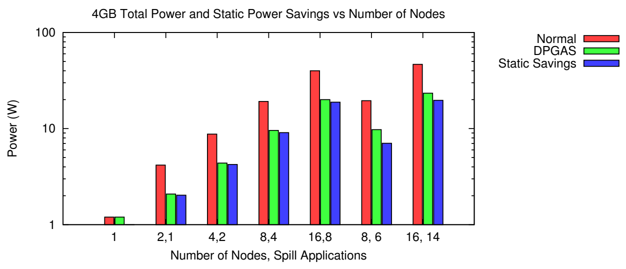

Figure 4. Total Power and Power Savings for Nodes With 4 GB DRAM

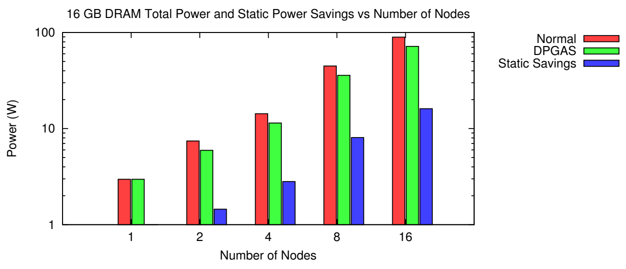

Figure 5. Total Power and Power Savings for Nodes With 16 GB DRAM

provisioning each DPGAS-enabled node with 4 GB instead of an 8 GB overprovisioned server, and for the 16 GB case, we assume that power savings are calculated in relation to a "normal" 20 GB overprovisioned server. Note that in the "memory blade" case, there are two receive nodes with an extra 4 GB of memory, leading to slightly higher total power.

The power savings are illustrated with respect to the total power (including power for activations, read, write, and termination commands) to give a proper sense of the total power consumed versus the static power that can be saved by provisioning fewer DIMMs. For the 4 GB case, each server receives half as much memory, so power savings are between 2 and 19 Watts or 47 to 49% for the normal spill/receive tests. For the 16 GB case, the overall percentage of savings is lower since we assume overprovisioning means adding only one more similar (4 GB) DIMM rather than adding multiple DIMMs. The savings for the 16 GB tests is between 1.5 and 16 Watts or 18% to 20%.

The memory blade experiments show similar savings for high-load environments. Compared to the medium load experiments (with an equal number of spill/receive nodes), the memory blade tests reduce static power by 36% and 42%, compared to 47% for the medium load simulations.

It should also be noted that removing DIMMs from the system can increase the overall system power if this reduction increases the system utilization. For instance, the HP Power Advisor Calculator [18] was used to build a 10,000 core data center with the similar DRAM characteristics to our experiments and based on the Proliant DL160 G6 server blade running at 60% utilization. For the 16 GB case, removing 4 GB of DRAM from half of the blades would result in a power savings of 3,540 Watts or 2.8%. However, if removing DIMMs increased utilization (CPU and memory) by 5%, the power cost would *increase* by 3,040 Watts or 2.5%. It is unlikely that our approach would increase CPU utilization, since modern operating systems are already geared toward hiding the latency of accesses to storage devices like DRAM and hard disks but more comprehensive experiments are needed to validate the effect of DPGAS on overall system utilization and accompanying power reductions.

C. Network Latency

In addition to using the HToE medium and DPGAS to virtualize DRAM DIMMs, HToE also helps to provide a low-latency interconnect solution that compares favorably with other mature solutions like RDMA and custom NUMA interconnects. To find the latency of an access to a remote DIMM, we must first calculate the "static" components of latency that are not modeled in our NS3 simulation model. The latency values for the related Ethernet and memory subsystem components were obtained from statistics from other studies [9] [19] [20] and from the place and route timing statistics for our HTEA implementation described in earlier work [1]. An overview is presented in Table III for convenience. The latency through the HTEA pipeline varies based on the type of packet (HT request or response) that is processed, but here we present the average latency based on an average processing time of six cycles at 125 MHz.

Table III
LATENCY NUMBERS USED FOR EVALUATION OF PERFORMANCE PENALTIES

Component	**Latency (ns)**
AMD Northbridge	40
On-chip memory access	60
Heidelberg HT Cave Device	45
HTEA	48
10 Gbps Ethernet MAC	500
10 Gbps Ethernet Switch	200
Average Component Delay	893
Measured Transmission and Buffering Delay (NS3)	185 - 939

Using the values from Table III, the performance penalty of a remote memory access can be calculated using the formula:

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

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

Figure 6 shows the total average one-way latency using

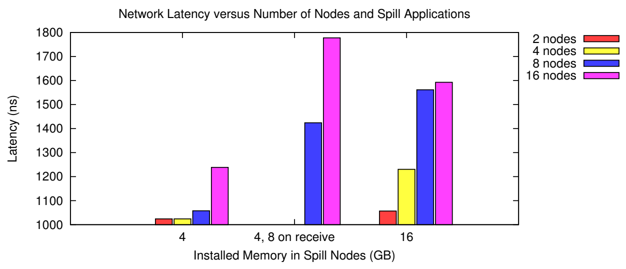

Figure 6. HW Latency for HToE Remote Operation

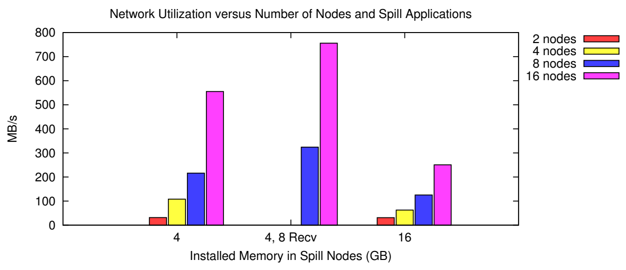

Figure 7. Network Utilization with DPGAS

the NS3 transmission latency values for $t_{transmit}$ and the average component delay in Table III for the other delay values. Note that we include the 200 ns switching delay [21] since NS-3 does not accurately model all the internal components of a data center-capable Ethernet switch. This addition likely overestimates the effect switching delay has on latency.

The one-way latency varies from 1024 ns to 1238 ns for the 4 GB simulations with six synthetic applications on each node and from 1057 ns to 1593 ns for the 16 GB simulations with sixteen applications on each node. Also, the addition of more spilling nodes doubles the latency of simulations with 8 and 16 nodes to 1478 and 1832 nanoseconds.

Although the overall latency increases due to buffering delay, latency still remains low enough that remote, virtualized DRAM can be accessed without causing noticeable delay to applications. The latency penalties reported here compare favorably to other technologies, including the fastest reported MPI latency, at 1.2 μs [22], RDMA (low μs range), and disk latency, which ranges from 6 to 13 ms for standard rotational drives [23].

The one-way latency can also be used to determine the total latency of a read request that receives a response:

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

Using these conservative values, we arrive at 2,242 ns for the read latency of a cache line.

D. Network Utilization

In addition to tracking packet latency, it is also important to analyze the overall impact of using HToE on a shared 10 Gbps Ethernet link. Figure 7 shows the link utilization in MB per second during the simulation time when DPGAS was being used to spill memory requests to remote nodes. Utilization varies from 31.3 MB/s for the two-node, 4 GB test case to 756 MB/s for the heavily loaded test case with 16 nodes and 14 nodes spilling to the two “memory blade” nodes. The 16 GB test cases show similar utilization to the normal 4 GB test cases, with utilization ranging from 31.3 to 250.65 MB/s. Although it would be expected that utilization would be higher for larger numbers of spilling applications, the synthetic benchmarks in the 16 GB tests transferred more data but over a much larger interval of time (about 4 seconds of real-world time versus 2 seconds in the 4 GB tests).

The addition of more nodes and applications per node increases the utilization, but the 10,000 to 20,000 cycle intervals between memory requests keep packets spaced out for medium load cases, reducing link utilization. The memory blade tests for 8 and 16 nodes show a dramatic increase in the amount of traffic with utilizations of 555 and 756 MB/s, respectively. For the sixteen node case, this translates to a utilization of more than 6 Gbps for the combined traffic of 56 applications sending HToE requests in the same period of time (a relatively heavy workload). This high utilization indicates that large memory blades may need to be placed on a separate 10 Gbps link to prevent delays for other network traffic.

VII. Discussion

Our simulations demonstrate a distinct point in time when multiple applications have overlapping requests to spill memory via DPGAS. In real-world situations multiple servers may experience peak workload at the same time, but we expect that our simulation setup, specifically for the memory blade tests, represents a heavy loading or worst-case scenario. While our simulations do not currently incorporate a full system performance model, DPGAS accesses should have much better performance than accesses to disk.

Additionally, our simulations did not take into account other existing optimizations for sharing memory such as page migration. This penalizes our results because pages that are most commonly used could be swapped into local memory, reducing the number of remote accesses. On the other hand, our simulations also did not take into account the HT transaction limitations (based on SrcTag and UnitIDs). This omission likely would increase the remote access latency in heavily loaded situations.

Finally, we note that there are several unsolved issues with using HT over Ethernet, mostly related to issues such as flow control. HyperTransport and similar on-chip interconnects are designed for fast point-to-point serial communication, so encapsulating HyperTransport packets into Ethernet will likely require additional buffering, protocols, and optimization to provide an effective implementation of HToE.

VIII. Related Work

An evaluation of power trends in the data center was conducted in [24], and a design for a memory blade approach to sharing memory, disaggregated memory, was discussed in [17]. The disaggregated memory approach has very similar goals to our work, but their implementation is different in that they focus on creating additional memory blades to increase available memory bandwidth and to enable coherent sharing between client nodes. Our work is focused on using existing DRAM resources more efficiently to reduce overprovisioning. While disaggregated memory provides better bandwidth than DPGAS, it may also prove cost prohibitive for small to medium data centers.

Our approach is similar to RDMA-based approaches, although DPGAS is focused on reducing registration overhead and supporting more fine-grained access. One company that has approached using RDMA for memory virtualization is RNA Networks, whose Memory Virtualization Platform uses RDMA with Infiniband or Ethernet to allow high-bandwidth and low-latency memory sharing [4]. This technique also helps to restrict memory power by reducing overprovisioning.

Other higher-level approaches have addressed operating system support and data center level support for reducing DRAM power. At the operating system level, Tolentino [25] has suggested a software-driven mechanism using control theory to limit application working sets at the operating system level and reduce the need for DRAM overprovisioning. Additionally, many researchers have proposed the concept of ensemble-level power management [26] to manage power at the server enclosure levels

IX. Conclusion

In this paper we have presented a fine-grained simulation infrastructure that was used to investigate the latency, bandwidth, and power characteristics of Dynamic Partitioned Global Address Spaces and its underlying hardware implementation, HyperTransport over Ethernet.

We have shown how DPGAS can be used to reduce DRAM overprovisioning in the data center and subsequently how to reduce static DRAM power. Additionally, we demonstrated that HToE is a low-latency encapsulation method that can be easily incorporated into existing data centers. Results from our simulations also provided insight into specific use cases and their effects on the link utilization of a standard 10 Gbps Ethernet switch.

In our future work we plan to further define the complete implementation of HToE, and we plan to incorporate support for tracking system-level performance and using optimizations such as page migration.

References

[1] J. Young, F. Silla, S. Yalamanchili, and J. Duato, "A hypertransport-enabled global memory model for improved memory efficiency," 2009, http://www.ub.uni-heidelberg.de/archiv/9796.

[2] U. Hoelzle and L. A. Barroso, *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan and Claypool Publishers, 2009.

[3] C. Lefurgy *et al.*, "Energy management for commercial servers," *Computer*, vol. 36, no. 12, pp. 39–48, 2003.

[4] "Rna networks memory virtualization for the data center," *White Paper*, 2008, http://www.rnanetworks.com/memory-virtualization.

[5] "An introduction to the intel quickpath interconnect," http://www.intel.com/technology/quickpath/introduction.pdf.

[6] "Hypertransport specification, 3.10b," 2009, http://www.hypertransport.org.

[7] "Fibre channel over ethernet final standard," http://www.fcoe.com.

[8] J. Nieplocha *et al.*, "Global arrays: a portable "shared-memory" programming model for distributed memory computers," in *Supercomputing '94*. New York, NY, USA: ACM, 1994, pp. 340–349.

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

[10] U. Brüning, "The htx board: The universal htx test platform," http://www.hypertransport.org/members/u_of_man/htx_board_data_sheet_UoH.pdf.

[11] "Ns-3 network simulator project," http://www.nsnam.org/.

[12] D. Wang *et al.*, "Dramsim: a memory system simulator," *SIGARCH Comput. Archit. News*, vol. 33, no. 4, pp. 100–107, 2005.

[13] *BIOS and Kernel Developer's Guide (BKDG) For AMD Family 11h Processors*, 2008, http://support.amd.com.

[14] A. Jaleel, "Memory characterization of workloads using instrumentation-driven simulation: A pin-based memory characterization of the spec cpu2000 and spec cpu2006 benchmark suites," *VSSAD Technical Report 2007*, 2007, http://www.glue.umd.edu/~ajaleel/workload/.

[15] S. Chalal and T. Glasgow, "Memory sizing for server virtualization," 2007, http://communities.intel.com/docs/.

[16] "Micron ddr2 part catalogs - twindie 4 gb," 2010, http://www.micron.com/products/dram/ddr2/.

[17] K. Lim *et al.*, "Disaggregated memory for expansion and sharing in blade servers," *SIGARCH Comput. Archit. News*, vol. 37, no. 3, pp. 267–278, 2009.

[18] "Hp power advisor utility: a tool for estimating power requirements for hp proliant server systems," 2010, http://h20000.www2.hp.com/bc/docs/support/SupportManual/c01861599/c01861599.pdf.

[19] 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.

[20] "Intel 82541er gigabit ethernet controller," http://download.intel.com.

[21] "Quadrics qs ten g for hpc interconnect product family," 2008, http://www.quadrics.com/.

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

[23] "Storagereview.com drive performance resource center," 2010, http://www.storagereview.com/.

[24] K. Lim *et al.*, "Understanding and designing new server architectures for emerging warehouse-computing environments," in *ISCA '08*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 315–326.

[25] M. E. Tolentino *et al.*, "Memory miser: Improving main memory energy efficiency in servers," *IEEE Trans. Comput.*, vol. 58, no. 3, pp. 336–350, 2009.

[26] P. Ranganathan *et al.*, "Ensemble-level power management for dense blade servers," in *ISCA '06*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 66–77.