# DYNAMIC PARTITIONED GLOBAL ADDRESS SPACES FOR HIGH-EFFICIENCY COMPUTING BY JEFFREY YOUNG

A Thesis
Presented to
The Academic Faculty

by

Jeffrey Young

In Partial Fulfillment
of the Requirements for the
Masters Degree in the
School of Electrical and Computer Engineering

Georgia Institute of Technology
December 2008

# ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Sudhakar Yalamanchili, for his support and advice in helping to bring this thesis to its completion and my committee members, Dr. Riley and Dr. Schimmel for their helpful comments. Also, I would like to thank my parents, Robin and Sybil, and my brother, Chris, for their continued support of my graduate studies. I also would like to thank my wife, Jessica, for all her understanding and assistance, without which this thesis would not have been possible.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

## LIST OF LISTINGS

# SUMMARY

The current trend of ever larger clusters and data centers has coincided with a dramatic increase in the cost and power of these installations. While many efficiency improvements have focused on processor power and cooling costs, reducing the cost and power consumption of high-performance memory has mostly been overlooked. This thesis proposes a new address translation model called Dynamic Partitioned Global Address Space (DPGAS) that extends the ideas of NUMA and software-based approaches to create a high-performance hardware model that can be used to reduce the overall cost and power of memory in larger server installations. A memory model and hardware implementation of DPGAS is developed, and simulations of memory-intensive workloads are used to show potential cost and power reductions when DPGAS is integrated into a server environment.

# CHAPTER I

# INTRODUCTION

Current trends in today's multi-core processors and cluster-based 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 the QuickPath Interconnect (QPI) has dramatically reduced the hardware cost for remote memory accesses. Both of these advances enable the creation of global non-coherent shared memory systems that have previously been commercially available and viable only in high-end supercomputers. This thesis introduces a new type of non-coherent shared memory system called a Dynamic Partitioned Global Address Space (DPGAS), implements a hardware component to support DPGAS, and evaluates how DPGAS can reduce memory cost and power requirements for servers.

## *1.1 Overview of Trends*

While this thesis focuses on an initial hardware implementation of DPGAS and its effects on reducing memory cost and power in servers, another contribution of this work is to evaluate the trends that motivate the use of DPGAS and that are important to consider for any kind of memory and network integration. These important trends are outlined briefly here and are discussed in more detail in Chapter 2.

While network latencies have dropped, and network interfaces have been moved closer to main memory and processors, multi-core-based systems have increased the pressure on existing DRAM memory as the number of cores per chip grows. Recent projections [33] indicate that memory pin bandwidth is likely to drop as more transistors are devoted to increasing performance with more processor cores. Reduced memory pin bandwidth can also be related to its effects on the trend of increasing virtualization of applications. As the number of cores per chip increases, processors can support an increased number of instructions from concurrent virtual machines (VMs), but each of these virtual machines will compete for a limited amount of memory bandwidth [57], causing memory pressure. This memory pressure can lead to performance imbalances between how fast each multi-core CPU can process data and how fast data can be transferred from off-chip DRAM. While many servers [29] can use software NUMA (Non-Uniform Memory Access) solutions to share unused memory bandwidth with adjacent processors, the complexity and scalability of NUMA-based solutions and the cost of server memory [26] often lead to the usage of virtual memory swapped in from associated hard drives to compensate for a lack of memory bandwidth.

In addition, the cost of high-performance DRAM for servers provides another incentive to efficiently share and utilize memory not only on each server rack but also between racks. Studies have suggested that many servers under normal workloads use on average about 1 Gigabyte of installed DRAM but also that physical memory requirements can vary greatly from that average amount [8]. While memory prices have dropped due to technology scaling, more cores will lead to a need for either higher-density (and more expensive) DIMMs or better memory sharing. The usage of a large non-coherent shared memory space would require fewer and smaller DIMMs to handle the same workloads, reducing system cost and Total Cost of Ownership overall due to reduced cooling requirements. This requirement for an efficient but dynamic shared memory mechanism lends itself to extensions of existing partitioned global address space (PGAS) models.

Current PGAS implementations focus on software level abstractions to share memory and are focused on data sharing for large message-passing based systems, such as those traditionally served by parallel programming languages like MPI. X10 [9], UPC [14], and Gasnet [5] are examples of existing PGAS environments that either operate at the software level or require low level implementations that must be ported for different hardware. These PGAS languages offer the ability for a programmer to define variables that are either considered local or shared, meaning that the variable may be stored in remote memory and accessed by multiple nodes. However, these existing PGAS implementations are mainly focused on programmer productivity at a higher level rather than efficient address translation and address space mapping at a low-level. By extending the functionality of these existing implementations to the hardware level, very fast and efficient address translations could be used to solve the previously discussed performance-related problems: 1) a future lack of memory bandwidth to local DRAM and 2) the need for simple but efficient mechanisms for sharing memory beyond the boundary of a server blade to reduce overall need for more memory. The extensions to standard NUMA and PGAS environments that provide for efficient address space management form the basis for the DPGAS model, which is implemented and evaluated in this thesis.

## *1.2 Outline*

This thesis exploits the availability of low-latency memory controllers that are tightly integrated with network interfaces (such as HyperTransport) to support a global, non-coherent 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 network to minimize the overhead for remote memory accesses and to allow for fast dynamic changes in address mappings. The feasibility of using this address space management to reduce server cost and power is evaluated using memory traces drawn from typical memory-intensive applications and a cost and power evaluation of four different server environments. This thesis will demonstrate that significant cost and power savings can be made by using DPGAS while also preserving the low-latency characteristics of remote memory accesses.

This thesis presents the following specific concepts:

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 evaluation of the usage of DPGAS with memory-intensive workloads to reduce overall power and cost in several different server environments

Chapter 2 investigates the trends that make DPGAS a viable choice for reducing cost and power in servers, and Chapter 3 defines the architecture and memory models for a DPGAS environment. Additionally, a hardware implementation of DPGAS, the HyperTransport over Ethernet bridge, is described in addition to an OS-level memory allocation algorithm that is used with the analytical evaluation of DPGAS in this thesis. Chapter 4 provides details on the memory-intensive workloads and page table simulation that are used to evaluate the cost and power savings of using DPGAS memory allocation, while Chapter 5 demonstrates the cost and power reductions that DPGAS can provide for different server environments via analytical models. Chapter 6 evaluates related work in the areas of PGAS, and cost- and power-efficient designs for memory and servers.

# CHAPTER II

# ENERGY CONSUMPTION AND COST TRENDS

## *2.1 Server Trends*

Several fundamental trends are shaping the large-scale multi-core systems space, specifically in the design of the memory subsystem. Memory latencies have remained relatively stable while memory bandwidth is increased to mask the effects of the "memory wall" [7] [57] [47]. As shown in Figures 1 and 2, memory latencies (for a column read) across several DRAM technologies remain constant while memory bandwidth has increased dramatically. With the exception of FB-DIMM memory, most standard memory technologies are focused on high-bandwidth memories and memory controllers (MCs) with large numbers of pins to enable high-bandwidth transfers.

However, at the chip level, ITRS projections for device density and number of pins show an increasing ratio of cores to pin bandwidth across successive technology generations for future chip multiprocessors (CMPs) [33]. Coupled with the migration of memory controllers on die, the available DRAM memory bandwidth per core on a single die will continue to decrease.

Compensating for this drop in DRAM bandwidth via commensurate increases in depth and size of on-chip cache hierarchies is not an option since increases in per-chip core count will compete with SRAM caches for small increases in die area across technology generations. Thus, in the absence of architectural or algorithmic innovations, performance scaling will be stymied due to lack of memory bandwidth.

## *2.2 Application Memory Footprints and Virtualization*

Concurrently, emergent applications and trends continue to increase memory pressure on CMPs. Modern and future data-intensive applications from scientific, enterprise, and database domains have produced applications with large, time-varying memory footprints and working sets as well as increased demands for memory bandwidth. For example, several of the

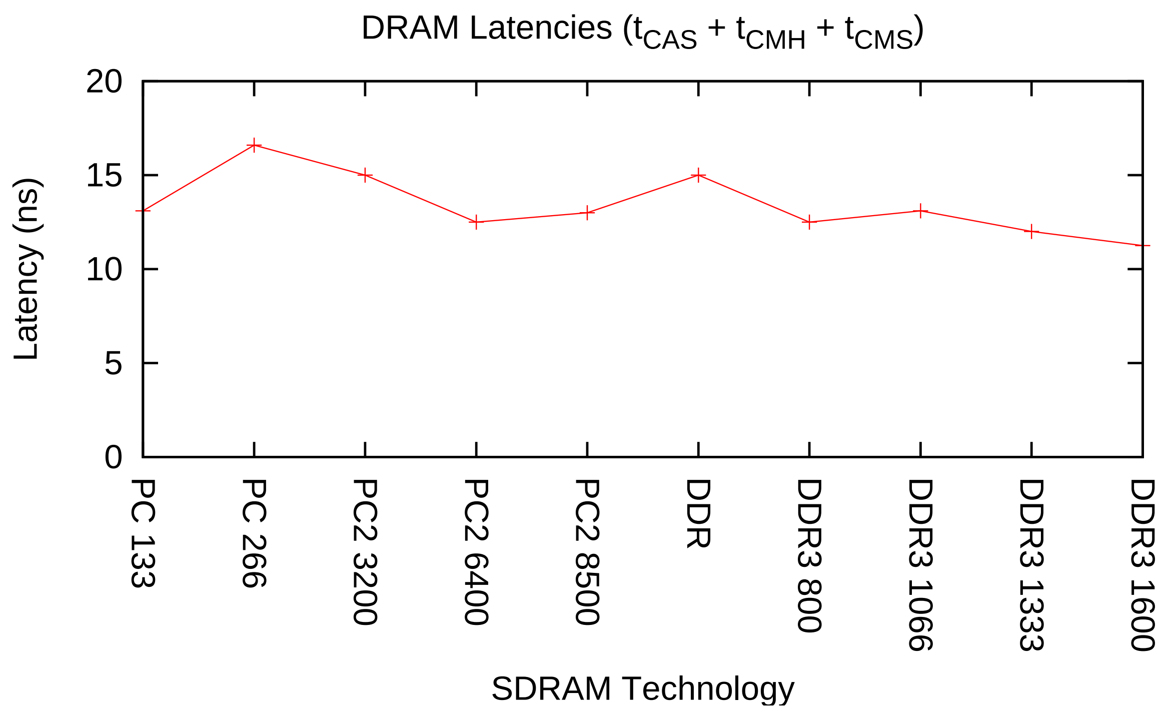


**Figure 1:** SDRAM Latency Trends

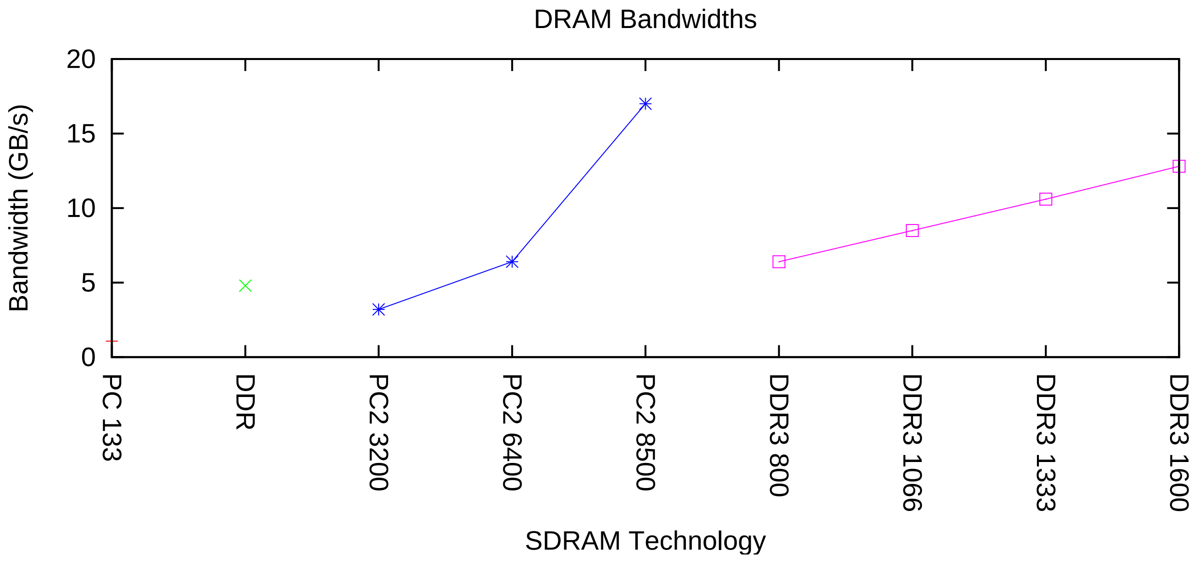


**Figure 2:** SDRAM Bandwidth Trends

SPEC2006 integer and floating point benchmarks have memory footprints that exceed 1 GB and average 900 MB [24]. Furthermore, several of these benchmarks have memory demands that vary quite a bit over time in terms of both the allocated memory and working set size [20]. A 2007 study of server workloads across approximately 3,000 machines found that about 50% of the applications have footprints that exceeded a gigabyte of memory [8]. Moreover, several studies of grid computing [39] and scientific computing [38] have confirmed the time-varying nature of application needs, meaning that application needs for memory are likely to be either periodic or widely varying depending on whether jobs are batched (as with supercomputers) or submitted on demand (as in cloud or grid computing).

Server consolidation via virtualization has increased by a multiplicative factor the memory demand of a single server. Thus, for future CMPs we see increasing demand for memory bandwidth in the presence of decreasing availability of memory bandwidth leading to significant increase in *memory pressure* on a single node or server.

To better illustrate this point, we note that memory pressure occurs when applications either are 1) consolidated onto fewer servers using virtualization or other techniques or 2) have such large memory requirements (footprints) that the application cannot be contained in physical memory.

While virtual memory and virtual machine allocation and migration are used to help mask the lack of physical memory on a server, performance penalties may increase for a particular application that has a large memory footprint for only a short portion of its runtime. If memory could be easily shared across server boundaries for these bursts of intense memory requirement, the entire system utilization could be improved while reducing the need to overprovision less to meet worst-case demands.

## *2.3 Interconnect Trends*

At a higher level, interconnects for connecting clusters have also experienced dramatic improvements in terms of latency and bandwidth. The link and switch cut-through latencies of modern 10 Gigabit per second (Gbps) Ethernet and Infiniband switches and links are comparable to modern DRAM access latencies, and are dropping faster than DRAM latencies.

Figure 3 demonstrates the trends for MPI Ping-Pong header messages on various interconnect switches for the last 10 years. The latency of a small MPI message has dropped from milliseconds in the 1990s down to 1.2 microseconds today [44]. While Infiniband, Myrinet, and Quadrics all typically have lower latencies, it is useful to notice that 10 Gbps Ethernet switches exist that have cut-through latencies of 200 ns for small messages at the hardware layer. Compared to an average latency of 10-15 ns for a DRAM memory read or write operation (excluding precharge and other setup) [45], interconnect latencies have dropped dramatically and at the hardware level are rapidly approaching the latency of a standard memory operation.

Current architectural trends also include the migration of functionality on chip including memory controllers as well as high-speed interconnects such as AMD's HyperTransport (HT) and Intel's QuickPath (QPI). AMD's HyperTransport interconnect provides for both coherent as well as non-coherent links and shared memory operation based on a broadcast protocol. Two and four socket HT-based single-server coherent shared memory systems have been commonplace for quite some time. While a specification for Intel's QuickPath has not been released, several trade publications point to similar physical characteristics [34] [49].

The hardware distance from the wire to the memory controller through the local cross-bar is very short, significantly reducing remote access latencies for non-uniform access memory (NUMA) models that extend across servers. However, HT and QPI are not switched interconnects. Therefore we either will need a customized inter-server interconnect (undesirable) or should make use of commodity interconnects. The proliferation of 10 Gbps and 40 Gbps technology and the expectations for 100 Gbps technologies motivate the use of Ethernet or Infiniband for remote accesses. Architecturally, cores now have access to memory controllers beyond the local server and therefore increased memory bandwidth in a demand-driven fashion. Effectively, the high interconnect bandwidths (the HT specification supports up to 40 Gbps in bidirectional bandwidth) are translated into memory bandwidth via access to remote controllers. This is managed in a demand-driven fashion.

Thus, with hardware paths establishing the physical latency properties, we need abstractions that can be employed by operating systems, middleware, and compilers to manage this

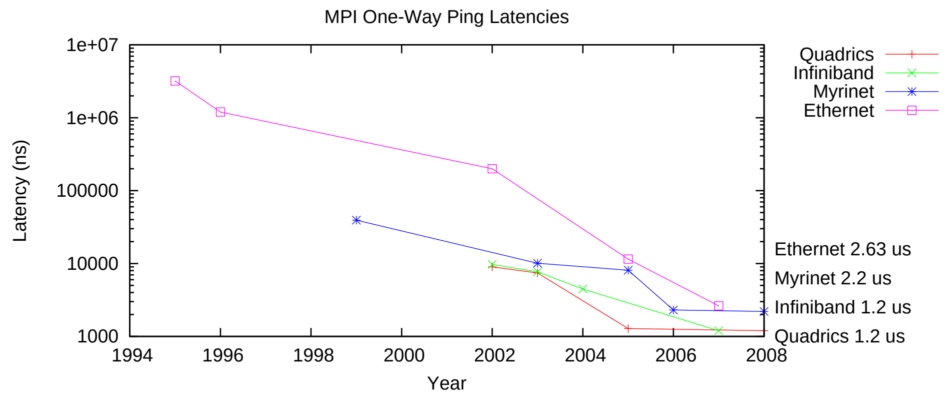


**Figure 3:** MPI Ping-Pong Latency Trends for the Last Decade

global, non-coherent shared memory system. These abstractions are described as part of the DPGAS model in Chapter 3.

## *2.4 Cost and Power Trends*

While the price of DRAM memory continually shifts due to the effects of Moore's Law, the price of commodity DRAM still does not scale linearly with increasing DIMM density, especially for higher-density memory chips. Several other studies [8] [37] have shown that fully populating a server can result in a nonlinear scaling in cost for high-density DIMMs. Figures 4(a) and 5(a) show memory cost trends for our low- and high-end servers that are used to evaluate the effects of DPGAS. For the low-end server in Figure 4(a), memory cost scales linearly due to the use of commodity-priced DIMMs. To fully populate this server would cost about $2,600. However, for the high-end server in 5(a), the cost for the 8 GB DIMMs grows at a rate that is at least two times the linear trend for smaller DIMMs. This is due to the added complexity and density of these chips and to manufacturing scaling—since 8 GB DIMMs are currently considered high-end, fewer companies have fully scaled out their manufacturing process to make these DIMMs. The progression of Moore's Law and manufacturing trends ensures that eventually these high-end DIMMs will be supplanted by 16 GB and 32 GB DIMMs, and the cost curve for 8 GB DIMMs will reflect that of the current smaller density chips. To fully populate this eight-core server with 512 GB of DRAM would cost an astonishing $83,900, a large sum considering the server's base cost of about $48,000. This figure also shows that if we were able to reduce overall server memory requirements from 512 GB to 384 GB (a decrease of 25%) through efficient memory usage, we would be able to save about $21,000 per server in memory costs alone.

Further, the energy provisioning, consumption, and cooling costs of large-scale data centers have become a major challenge. The memory system alone has been projected to account for anywhere from 15-30% of these energy costs, which again are amplified when provisioning the memory/server for peak demand. Additionally, power consumption also increases with larger-density memories, and can require just as much power as processors in large configurations [37]. This has in part led to proposals such as the use of memory servers [41] to provide a shared global pool of physical memory that can be used to smooth out peak demands. For our two server configurations in Figures 4(b) and 5(b), we see that larger-density DIMMs scale linearly in terms of power consumption, especially in the high-end case. The usage of commodity parts in the low-end server leads to more uniform requirements between different DIMM sizes. This also reinforces the notion that memory selection for high-end servers can often be a compromise between cost and power usage.

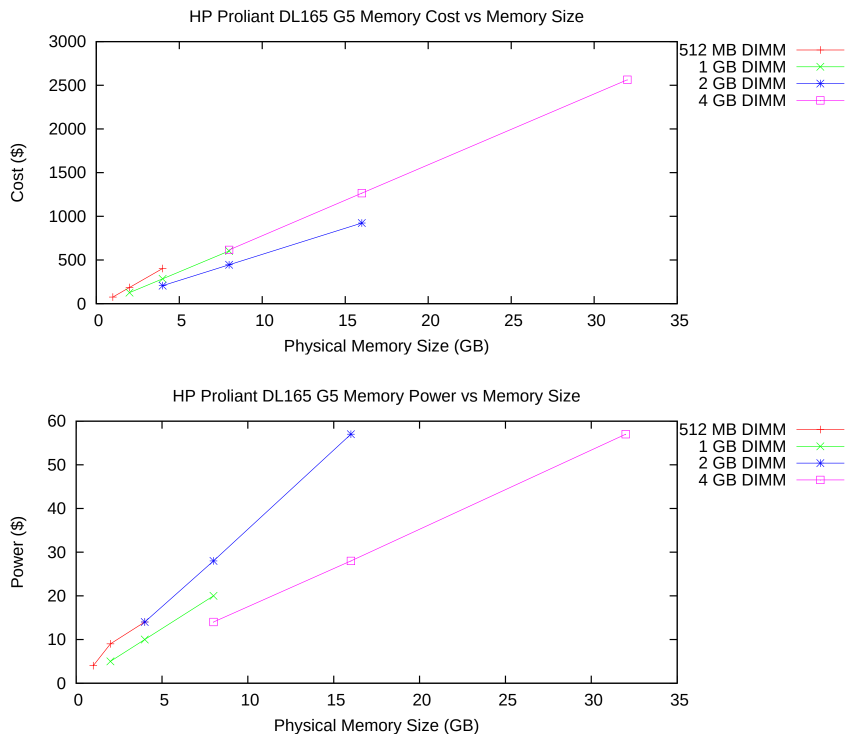
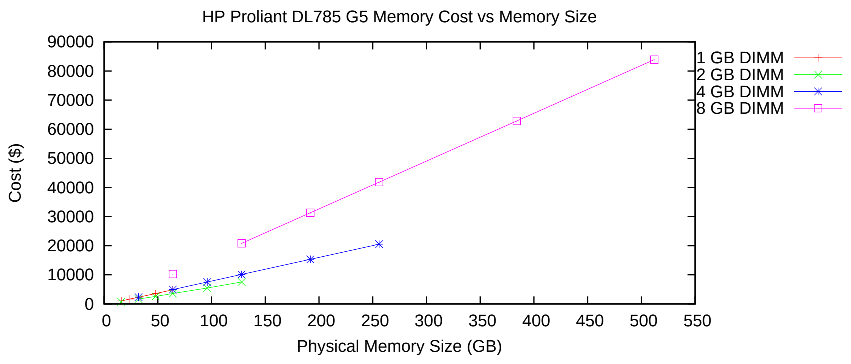


**Figure 4:** Memory Cost and Power Trends for Proliant DL165 G5



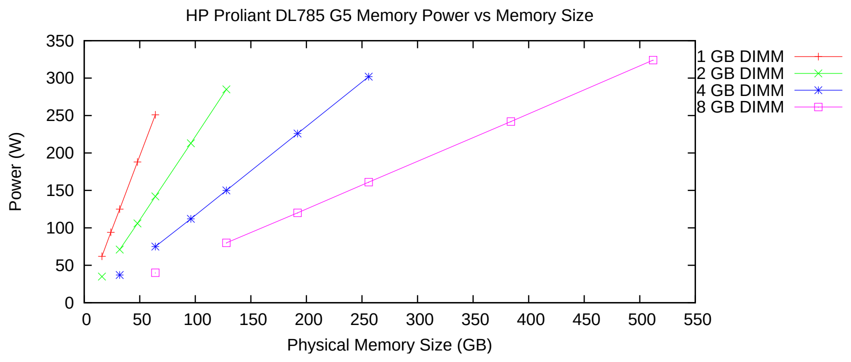


**Figure 5:** Memory Cost and Power Trends for Proliant DL785 G5

Finally, the architecture of servers presents several practical constraints that interact with and amplify the preceding trends. For example, the memory reach of a server is currently limited to the on-server memory. While NUMA techniques can be used to share memory, they do not offer any additional controls over physical address space management. Management of the physical address space can be useful for several reasons, which are discussed more in Section 3.2.

Both of these trends, power and cost, point to a need for using lower-density and lower-cost memory DIMMs even in large server configurations that may require higher performance. Servers are often overprovisioned in terms of memory to keep from scaling out with more lower-density servers. Overprovisioning increases the total amount of memory that is idle, while other servers may be overutilized but cannot easily request or share memory beyond their blade boundaries.

Consequently, we note that it is important to be able to dynamically provision and share memory across servers to meet instantaneous peak memory demands of applications or virtual machines executing on a single server. This leads to a much smaller physical memory footprint for the system and consequently lower dollar and energy costs. We propose memory sharing via tight coupling between the interconnection network and the local memory controllers. The result is the CMP cores have access to greater instantaneous memory bandwidth and larger amounts of physical memory in a demand-driven fashion. These trends and system architecture drivers lead us to conclude that:

**Thesis Statement**: By defining a global address space and hardware mechanism for sharing remote memory, we can allow applications with time-varying footprints to efficiently share memory across fewer numbers of servers and clusters while preserving performance and reducing overall power and cost usage.

# CHAPTER III

# A DYNAMIC PARTITIONED GLOBAL ADDRESS SPACE MODEL

This section describes extensions to a well-known PGAS model [9] to permit flexible, dynamic management of a physical address space, called Dynamic Partitioned Global Address Space (DPGAS). The two components of the DPGAS model are the architecture model and the memory model. Additionally, this section characterizes the hardware component that this thesis contributes to implement DPGAS, the HyperTransport over Ethernet bridge, and illustrates a simple OS-based memory allocation algorithm that is used in our first-order analysis of DPGAS.

## *3.1 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. An example of the latter includes AMD's HyperTransport protocol [30]. 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 [10]. 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 [1] or a custom interconnect such as those employed in high-end computing configurations by Cray [11]. In this case a get or put 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 put and get transactions from any core. Finally, the DPGAS model asserts that the lowest latency is achieved when the memory controller is tightly integrated with the Network Interface (NI), effectively minimizing the distance from the DRAM to the wire. While this integrated NI-MC is not available in today's architectures, migration of components like interconnects and memory controllers on chip seem to indicate that future architectures could feasibly contain an integrated NI-MC.

The architecture model is memory-centric in the following sense: Cores are becoming primitive architectural elements that are no longer the primary determinant of performance because 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.

## *3.2 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 224 partitions with a physical address space of

240 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. 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 non-coherent to allow for more 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 which may suffice for use with the DPGAS model.

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, a network that closely approximates the desired integrated NI-MC would be optimal. Being constrained by commodity parts, this study utilizes Gigabit Ethernet.

Once the DPGAS memory model is enabled, an application (or process) can now be allocated a physical address space that may span multiple partitions, i.e., local and remote partitions. Equivalently, the processes' physical address space is mapped to multiple memory controllers, and thus an application's virtual address space may map to physical pages distributed across local and remote partitions. There are a number of reasons to physically distribute an application's physical address space. The most intuitive one is for sharing where processes from an application may share physical pages by having portions of their virtual address spaces mapped to the same physical pages. No assumptions need to be made about how that shared space is managed, leaving open options for optimized management that are specific to the type of shared interactions, e.g., communication and shared libraries.

The set of physical pages allocated to a process can be static (compile-time) or dynamic (run-time). The nature of the page management changes in scalable systems due to the hierarchy of latencies necessitating optimizations that have little relevance in traditional operating systems, e.g., page placement. Physical partition and page allocation can affect the communication support that must be provided. For example, it may be necessary to maintain a list of remote partitions that can be accessed or information related to coherence/consistency management that may be maintained on a per-page basis. A likely candidate for inter-node communication of any dynamic changes in allocated physical pages is a simple Remote Procedure Call (RPC) that is executed infrequently.

To summarize, the DPGAS model specifies get/put transactions for accessing physically distributed pages of a process, and these transactions may have posted or non-posted semantics. The location of physical pages may be changed under compiler or operating system control, but the pages remain in a global 64-bit physical address space. All remote transactions are necessarily split phase.

## *3.3 DPGAS Implementation*

This thesis contributes a hardware component to the DPGAS model called the HyperTransport over Ethernet (HToE) bridge. The HToE bridge uses the on-board HyperTransport (HT) interconnect integrated with a 1 Gbps Ethernet Medium Access Control (MAC) into a

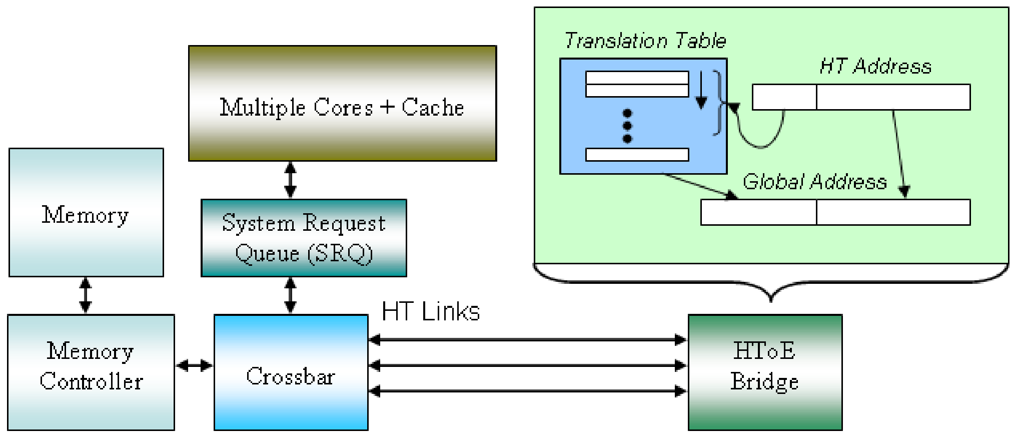


**Figure 6:** HToE Bridge with Opteron Memory Subsystem

Verilog component and is shown in relation to the other pieces of the Opteron memory subsystem in Figure 6. It incorporates an address translation table to translate local HT packets to global (64-bit) HT packets and an encapsulation mechanism to send these global packets over a system-level interconnect such as Ethernet or Infiniband. The HToE bridge fulfills the requirements of the DPGAS architecture and memory models described in Sections 3.1 and 3.2 by allowing for the implementation of a remote get/put operation.

HyperTransport was selected as a low-cost on-board interconnect due to its tight integration with the processor and memory using a crossbar for each processor connection instead of the previous Front Side Bus. HT was also chosen because Intel's QuickPath Interconnect has not currently been released and likely will not have an open-source specification. Gigabit Ethernet was selected due to current hardware availability and its ubiquity in clusters as a low-cost interconnect. While Ethernet is not geared toward the High Performance Computing (HPC) market but more toward widespread, low-cost deployment, the introduction of new standards for 40 Gigabit and 100 Gigabit Ethernet ensure that it will provide a low-cost competitive commodity interconnect. This thesis uses 1 Gbps Ethernet as a demonstration vehicle while recognizing that other implementations may realize encapsulation using other technologies such as 10 Gbps Ethernet, Infiniband, Myrinet, or Quadrics interconnects.

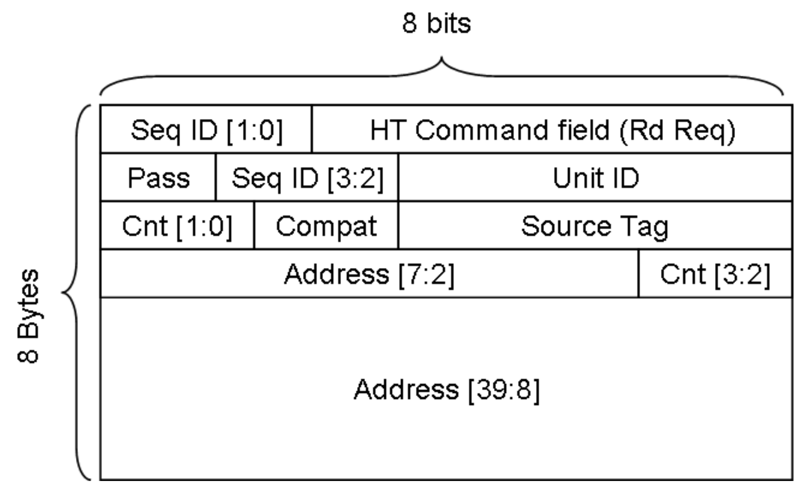


**Figure 7:** HT Read Request Packet Format

### 3.3.1 HyperTransport Overview

It is useful to first provide a simple overview of key operational attributes of HyperTransport. In its simplest instantiation, HT devices are connected via a point-to-point, one-dimensional topology anchored at one end by the host bridge. The host bridge implements the interface to the rest of the system. Data and control packets are transmitted over three classes of virtual channels: posted, non-posted, and response. HT device request packets travel upstream to a host bridge where they are either 1) routed upstream to a higher level device or main memory, or 2) routed back downstream to the target device. As an example, an HT read request command packet with 40-bit addressing (for pre-Barcelona CPUs) is shown in Figure 7.

Posted packets are typically associated with operations that don't require a response (such as some writes), and non-posted packets are used when a response packet is desired, either for a read or a write which may have a response "TargetDone" packet to indicate the write completed correctly. In addition, the HyperTransport specification defines flush and fence commands to help prioritize data as each virtual channel is multiplexed through the various HyperTransport interconnects. The HyperTransport specification also defines how packets in each virtual channel are prioritized when they are multiplexed over a common link. Basically, nonposted requests and responses can pass posted write requests if they have a certain bit field set. Additionally, HyperTransport usually specifies no ordering requirements for requests or responses unless their command packets have a certain field set that defines a strict ordering. Without strict ordering, requests and responses can pass each other when traversing HT links, assuming that there are enough credits (buffer space) available for them to be transmitted. Our model adheres to normal HT ordering and deadlock avoidance protocols.

The fields of the command packet, such as the ones in Figure 7 are used to specify options for the read transaction, and to preserve ordering and deadlock freedom. The most important fields for this example are the UnitID, SrcTag, SeqID, and address. 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 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 [30]. The current implementation in this thesis assumes that 40-bit addresses are used since this was the standard for Opteron processors during the first bridge implementation, but this can be easily changed to support 48- and 64-bit physical addresses. Additionally, the use of 40-bit physical addresses allows for backwards compatibility with previous versions of HT-enabled processors.

The HToE bridge implementation uses the University of Heidelberg's HyperTransport Verilog implementation [52] which implements an HT cave (end point) device. The application interface was retained to communicate between the local HT link and the HToE bridge. Figure 8 shows the stages of the HToE bridge.

### 3.3.2 HyperTransport over Ethernet - Address Translation and Ethernet Encapsulation

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 or Celoxica's RCHTX board [6] [51]. 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:

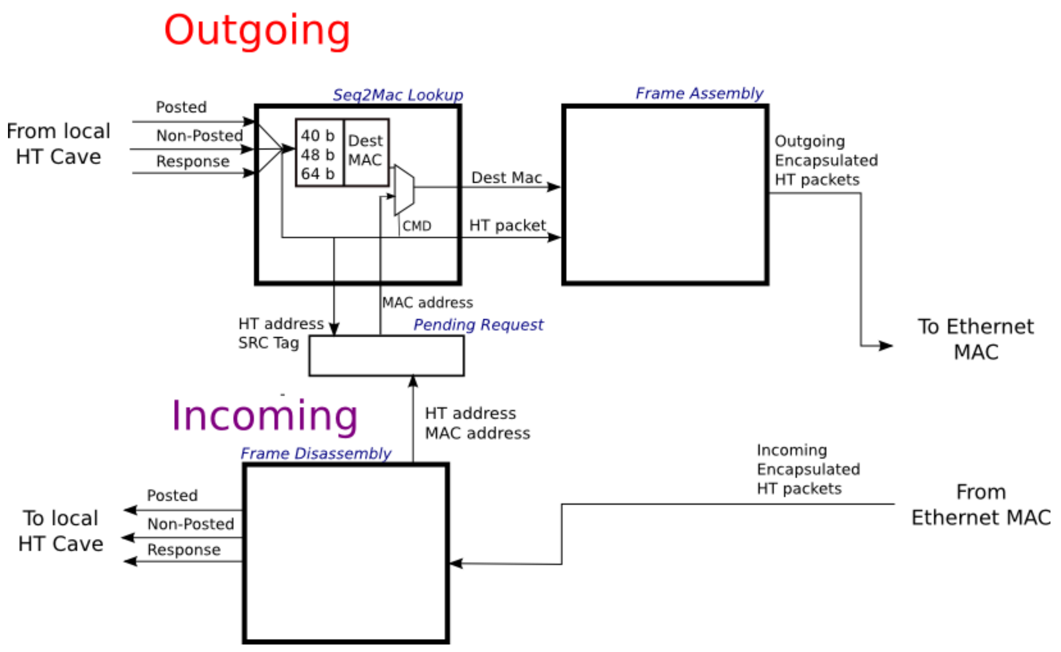


**Figure 8:** HToE Bridge Stages

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, i.e., all of the configuration registers in the DPGAS bridge have been loaded with mappings for destination addresses that map to an application's remote memory space. Now consider a parallel, shared memory 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 HyperTransport packet which 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

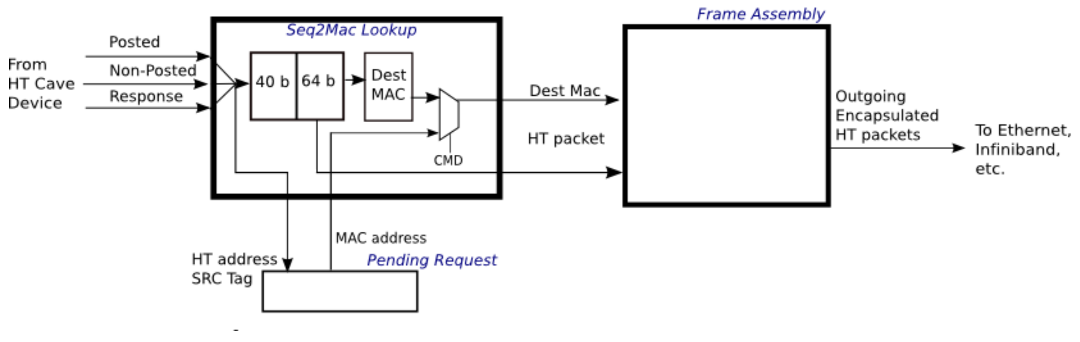


**Figure 9:** HToE Read Request Transmission

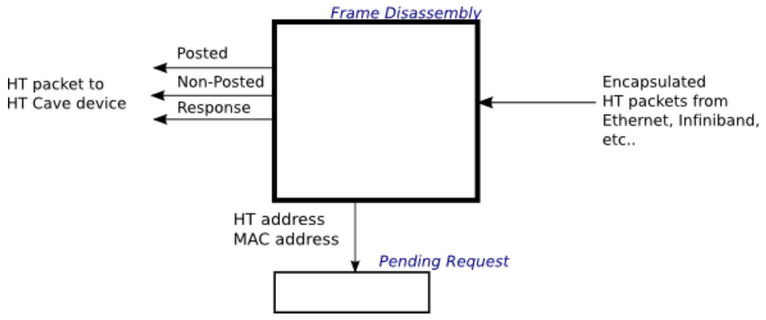

Frame Disassembly
HT packet to HT Cave device
Posted
Non-Posted
Response
Encapsulated HT packets from Ethernet, Infiniband, etc..
HT address MAC address
Pending Request

**Figure 10:** HToE Read Request Reception

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—for example, Infiniband. Thus, some efficiency was sacrificed for initial ease of implementation and for a cleaner, modular design.

The detailed transmit behavior of the HToE bridge for a read request to a remote partition is described by Figures 9 and 10.

First, the HT packet type is decoded into a request, response, or command packet in the module called Seq2Mac in Figure 9. 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 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 8 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.

Performance results for this model and the HToE bridge are investigated in more detail in secton 5.6.1.

## *3.4 Applications of DPGAS to Improve Memory Efficiency*

As the previous sections illustrate, DPGAS allows for the dynamic allocation of physical pages from a full 64-bit address space using simple get/put encapsulation. DPGAS is envisioned to work with a variety of operating systems–based techniques to manipulate the address mappings in the HToE bridge, but it also needs a memory allocation component in order to decide how remote memory is remapped using the HToE bridge. This section outlines the memory allocation technique evaluated in this thesis, percentage improvement. The pseudocode and usage of this algorithm is discussed in Section 5.1.1.

### 3.4.1 Memory Allocation at the Operating System Level

At a high level, nodes experiencing memory pressure need to request free memory that exists on remote nodes, and these remote nodes need to allocate memory fairly to all requesting nodes according to some basic metric. The initial memory allocation scheme defined for this thesis focuses on requesting and allocating memory from nodes that are closest in terms of number of hops (nearest neighbors) in order to keep remote access latency to a minimum. Assuming that the network uses a torus interconnect, each node would be limited to requesting and allocating memory to nodes that are at most one to two hops away. A request for memory can then be made by an overutilized node to a nearest neighbor for an interval of time using high-level communication like server RPCs. This allocation, or memory “lease”, would last for at least several milliseconds to seconds without further communication between the nodes. A requesting node can also send a message requesting a renewal of the lease for a specified number of lease periods as long as there is no other contention from other neighboring nodes. If at the end of the lease, the node hosting remote memory accesses decides that it needs its physical memory, it can reject new lease requests or send a message terminating existing lease renewals. Additionally, the node granting a lease can use a very simple LRU algorithm to allocate remote memory between several nearest neighbors contending for space.

The memory allocation algorithm can take several forms—the virtual memory manager on a node can allocate memory to remote nodes in a round-robin fashion, first-come first-served, or by least recently allocated, or it can reserve its local memory for anticipated memory-intensive tasks it might be running and reject all memory lease requests. This thesis focuses on a memory allocation scheme that helps improve overall system performance—percentage improvement. Percentage improvement tries to define how much memory is needed to gain a certain percentage improvement in application performance. For instance, if a local node has free memory, and it can improve the performance of Remote Node One by 10% by allocating an extra 2 GB of DRAM to Remote Node One’s application, but it can improve performance by 30% by allocating the same amount of memory to Remote Node Two, it may make sense to optimize memory usage to improve performance of the entire system rather than just performance on one node.

$$\%\, improvement = \frac{\%\, reduction\ in\ page\ faults}{amount\ of\ memory\ allocated}$$

This type of analysis does require some dynamic profiling, but each system could profile its own code and include a simple estimate with each memory lease request to allow the local allocator to decide how best to allocate memory. This memory allocation technique is used with the analytical models for evaluation of memory power and cost with DPGAS in Chapter 5.

# CHAPTER IV

# MODEL AND EVALUATION

## *4.1 Efficiency Model*

In order to determine DPGAS's effect in reducing power consumption and memory cost, two realistic server configurations were selected to test DPGAS memory allocation versus normal memory allocation. A high-end server configuration and a low-end Beowulf-capable server were selected to provide two different cost and power points and to evaluate how DPGAS affects each of these different server installations. The high-end server selected for this study was the HP Proliant DL785 G5, which has slots for 64 DIMMs and up to 512 GB of PC-5200 DRAM along with eight sockets capable of running quad-core Opterons. The low-end server used was HP's Proliant DL165 G5, which has two sockets and support for 32 GB of memory available in eight DIMMs. Both the Opteron and Xeon chips only support one hardware thread. Both servers were also only configured with one hard drive, hot-swappable in the high-end case and non-hot-swappable in the low-end case. Cost estimates were obtained from HP's business website [26] and from a third-party vendor [12]. Power statistics were generated using the HP Power Calculators with a 75% load factor [28]. The two server setups are described in more detail in Table 1.

**Table 1:** Server Configurations Used for Cost and Power Evaluation

| Model | CPU Cores | Maximum Physical Memory |
|---|---|---|
| HP Proliant DL785 G5 | 8 quad-core 2.4 GHz Opterons | 512 GB |
| HP Proliant DL165 G5 | 1 dual-core 3.0 Ghz Xeon | 32 GB |

The latency statistics for the HToE bridge component and related Ethernet and memory subsystem components were obtained from statistics from other studies [10] [52] [32] and from place and route timing statistics for our bridge implementation. An overview is presented in Table 2 and the bridge statistics are discussed more in Chapter 5. 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. 10 Gbps Ethernet numbers are shown in this table to demonstrate the expected performance with known latency numbers for newer Ethernet standards.

**Table 2:** Latency Numbers Used for Evaluation of Performance Penalties

| **Interconnect** | **Latency (ns)** |
|---|---|
| CPU to on-chip memory | 80 |
| Heidelberg HT Cave Device | 35 - 55 |
| HToE Bridge | 20 - 40 |
| 10 Gbps Ethernet MAC | 500 |
| 10 Gbps Ethernet Switch | 200 |

## *4.2 Evaluation Model*

The evaluation criteria for the DPGAS model consists of two experiments to evaluate two different situations where DPGAS might positively affect cost and power usage. The first evaluation looked at the case of server scale up, where servers are dramatically overprovisioned to provide better performance. Secondly, a large-scale case of scale out was evaluated, where more servers are added to relieve memory pressure that exists on a smaller number of servers. Current values for supercomputers are in the range of 1,024-4,096 processors [27], while data centers can vary from 1,000 to tens of thousands of processors in a typical server farm. For the scale up and scale out evaluations, the metrics used were page faults and overall cost and power usage, and DPGAS memory allocation was compared to normal memory allocation for three cases where each blade was either overprovisioned or underprovisioned in terms of how much memory was installed in the available DIMMs. Page swapping operations are representative of overall performance in the system, especially in the case of major page faults (pages that swap from memory to disk). In addition to cost and power analysis, the performance penalty of the HToE bridge was evaluated to determine the additional latency to access remote memory and optimizations that can improve performance. These evaluations are explored in Chapter 5; the rest of this chapter describes the tools and models used to perform the DPGAS cost and power analysis.

**Table 3:** Benchmarks Used for Evaluation

| Benchmark | Suite | Memory Footprint (MB) | Input Set |
|---|---|---|---|
| SSCA #2 | HPCS | 297 | 21 vertices |
| Transitive Closure | DIS | 275 | 6000 vertices, 3.6 million edges |
| MCF | SPEC 2006 | 1600 | ref |
| MILC | SPEC 2006 | 655 | ref |
| LBM | SPEC 2006 | 409 | ref |

### 4.2.1 Benchmarks and Simulation Model

Benchmarks were selected from both the High Performance Computing and enterprise arenas. The benchmarks include the HPCS Scalable Synthetic Compact Application benchmark for graph analysis [2], the Transitive Closure benchmark from the updated DIS Stressmark Suite [13], and three benchmarks from the SPEC CPU2006 [23] suite: the integer MCF benchmark and the floating point LBM and MILC benchmarks. The HPCS graph analysis benchmark implements four kernels that perform memory-intensive operations on a directed multigraph and has poor memory locality. The DIS Transitive Closure benchmark solves an all-pairs shortest path algorithm and was scaled up from its original input size due to technology improvements since its last update. The SPEC MCF benchmark deals with a combinatorial optimization problem applied to a mass-transit simulation. The SPEC floating point benchmarks, MILC and LBM, are both representative of large-scale simulations using floating point computations to simulate quantum chromodynamics and computational fluid dynamics with the Lattice Boltzman Method, respectively.

Memory footprints for the benchmarks were gained from other papers [20] and by using a simple script that called the Linux "ps" command to capture the resident physical memory usage under Linux. These memory footprints are shown in Table 3.

In order to demonstrate the effects of memory pressure from each application, a page table simulator was built to hash 64-bit virtual addresses from benchmark traces into simulated physical pages. Section 4.2.2 describes the trace generation in more detail. Page replacement used a simple clock-based LRU algorithm similar to what is used in the Linux kernel; statistics were gathered, including percentage of reads and writes, operations, and hits and misses; and replaced pages were tracked for each virtual address in the page table. Also, a warm-up period was used with each simulation to help discount the effects of compulsory misses.

Detailed per-page statistics were kept in order to better understand the memory access patterns for virtual memory and to help in evaluating which virtual addresses would be best suited for mappings to local physical pages as well as page migrations from remote to local physical pages. These statistics are not included as results in this work, but their potential usage for future work is described in Section 5.8.

### 4.2.2 Trace Generation

Trace generation was performed by instantiating a single-threaded subset of each benchmark inside the full-system simulator, Simics [42]. Simics models a base processor model, in this case an AMD Athlon 64 processor, and an installed operating system, in this case Fedora Core 5 Linux running the 2.6.15 kernel. Physical memory size is configurable at system startup and allows for simulation of SMP processors. This study used a single processor and single-threaded benchmarks to isolate the effects of each benchmark on the physical memory footprint. Additionally, single-threaded benchmarks represent the effects of either a single task running in a Virtual Machine on a server or a portion of a large multi-threaded application running in parallel on a cluster. It is expected that multi-threaded applications would require a similar amount of physical memory and possibly more due to sharing between threads, so a single-threaded application also represents the best case memory usage, assuming that enough physical memory is available.

Trace files of 2.1 billion memory instructions were collected by first profiling each benchmark with the gnu profiling tool, gprof [21], and Valgrind's dynamic heap allocation tool, Massif [46] [43]. After determining which function in a program was responsible for the largest percentage of computation time and approximately where the application allocated the most dynamic memory (i.e., the memory footprint was near its maximum value), a Simics "magic-break" instruction was used to enable trace capture starting at this point in the program. Insertion of the Simics breakpoint ensured that memory traces were pulled from the point when the application had the largest memory footprint and that memory traces captured a portion of the program that was computationally significant. Operating system traffic was included in each trace in order to incorporate the effects of the operating system. The overall memory footprint of each application reflects a relatively quiet run with minimal system processes running in that no other applications except normal system services were run during trace generation. This trace generation process reflects the fact that application performance is typically impacted by some kind of operating system traffic and that some small portion of page faults might be influenced by operating system services.

The inclusion of OS-related memory accesses resulted in almost 20% of the virtual addresses mapping to operating systems calls, typically related to instruction fetches. While 20% of the total memory accesses is a significant portion of the total trace, it also represents the standard overhead of a Linux operating system as measured in our Simics trace generation setup. The 2.1 billion memory instructions, which represented a point in the application with the maximum memory footprint, were then fed into the C++ simulator described in Section 4.2.1, with 100 million instructions being used as the warm-up period.

# CHAPTER V

# EVALUATION AND RESULTS

As discussed in Chapter 4.1, the evaluation of DPGAS contains two components: an evaluation of page fault improvements and memory power and cost reduction using DPGAS for four different server configurations and an evaluation of the performance penalties that remote memory accesses incur.

## *5.1 Page Fault Trends*

Figure 11 shows the total number of page faults for each memory size in the page table simulations for the five benchmarks that were tested. As the amount of physical memory is increased to be as large as the application's working set, the page fault rate drops to a static point where page faults that occur are most likely due to compulsory misses. This graph also illustrates the variation in worst-case memory requirements for each application. As discussed in Section 4.2.1, MCF requires about 1,600 MB of virtual memory to satisfy its working set, while the DIS Transitive Closure application only requires about 275 MB.

### 5.1.1 Memory Allocation for Experiments

Table 4 shows a more important metric for the memory allocation scheme used in this work—percentage improvement for each different physical memory size as first discussed in 3.4.1. For instance, allocating an extra 64 MB to the DIS application would result in an improvement of about 8.6% whereas allocating that same 64 MB to MILC would only result in an improvement of 0.21%. This higher reduction of page faults reflects that DIS has many more page faults when it is allocated a small amount of physical memory, whereas MILC performs similarly as its memory footprint is decreased. The results in Table 4 were used with percent improvement memory allocation in that remote allocations focused on servicing these higher-priority applications first to improve overall system performance while allowing for power and cost reductions by using less memory overall.

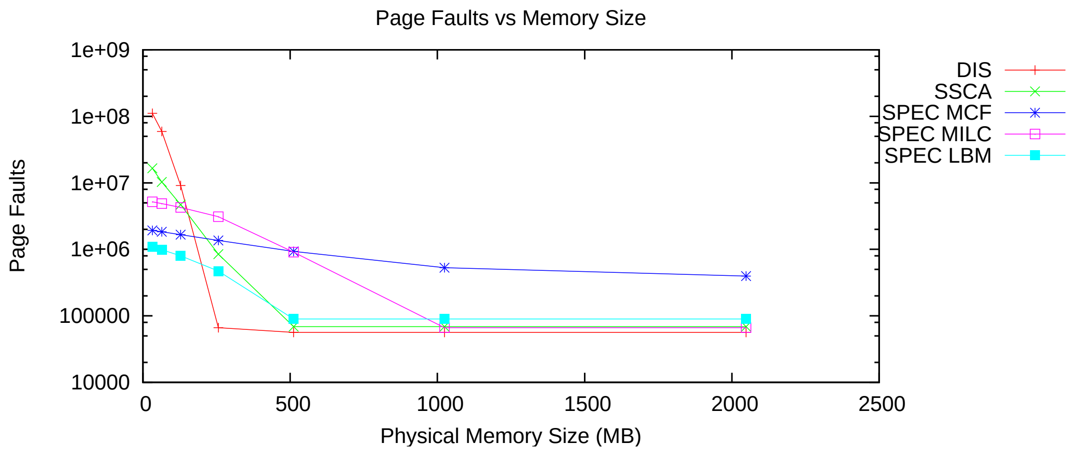


**Figure 11:** Page Fault Trends for Sample Benchmarks

Listing 1 shows the pseudocode used to allocate any free memory available with servers that implement DPGAS and a percent improvement memory allocation model. This model was implemented in the analysis of the HP server configurations that follow, but it does not preclude the usage of other memory allocation algorithms, such as slab and buddy allocation in Linux. Additionally, algorithms designed for shared memory and NUMA architectures like memory balancing [35] and Berkeley's Firehose [3] could be used with DPGAS.

The percent improvement algorithm assumes existing profile knowledge of applications' paging requirements, and is most likely a higher overhead algorithm for maximizing the performance of all applications in a given system. Other allocation algorithms may not be able to preserve the same level of application performance but still can provide the same memory cost and power savings when used with DPGAS, while requiring less knowledge of application memory profiles.

**Table 4:** Percentage Improvement For Benchmarks

| | **DIS** | **SSCA** | **MCF** | **MILC** | **LBM** |
|---|---|---|---|---|---|
| 64 | 2.7315 | 1.8972 | 0.1572 | 0.2022 | 0.3087 |
| 128 | 8.5684 | 1.8675 | 0.1659 | 0.2149 | 0.3673 |
| 256 | 106.7267 | 3.5460 | 0.1743 | 0.2958 | 0.5425 |
| 512 | 0.0685 | 4.4035 | 0.1791 | 0.9475 | 1.6614 |
| 1024 | 0.0000 | 0.0000 | 0.1483 | 2.4725 | 0.0000 |
| 2048 | 0.0000 | 0.0000 | 0.0331 | 0.0000 | 0.0000 |

**Listing 1:** Memory Allocation With Percent Improvement

```
While free memory is left
  For each VM workload
    For each server
      Reset winner and % threshhold
      for each VM running on that blade
        if % improvement for VM $\geq$ % improvement of other VMs
          set VM as winner and set % improvement as new threshhold
      if winner exists
          allocate memory via DPGAS to the winning VM
      else if no winners
          allocate memory locally as needed
```

## *5.2 Workload Models*

Earlier studies have suggested that virtual machines can be consolidated at a ratio of 15-20 VMs for a dual-socket quad-core system and 16 GB of memory [8]. Since the selected high-end server configuration has four quad-core processors, it stands to reason that four times as many VMs can be supported with 64 GB of DRAM. This scaling assumes that each VM only requires 0.6-1.5 GB, when in actuality, the same study suggested that a quarter of real systems used more than 3 GB of DRAM. However, the evaluation in this thesis focused only on workloads with maximum memory footprints from 0.3-1.6 GB, so this scaling factor was used to model a dual-core server supporting 15-20 VMs and a quad-core server supporting 60-80 VMs.

Table 5 shows the different workloads used for the scale up case with the low-end server. These workloads were arbitrarily chosen but do exhibit a varied workload as would be typical in a real server environment. These workloads were scaled up for the high-end server analysis and for the scale out analysis to multiple servers. In the scale up case, these workloads were scaled up by a factor of four, and for 250 servers, this workload was replicated 31-32 times to represent the maximum memory footprint for these servers.

**Table 5:** Workload Modeling for Low-End Server Case

| VM instances | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Blade Number** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
| DIS | 6 | 5 | 4 | 3 | 2 | 5 | 4 | 3 |
| SSCA | 5 | 4 | 3 | 2 | 6 | 3 | 3 | 4 |
| MCF | 4 | 3 | 2 | 6 | 5 | 3 | 2 | 5 |
| MILC | 3 | 2 | 6 | 5 | 4 | 2 | 4 | 2 |
| LBM | 2 | 6 | 5 | 4 | 3 | 4 | 3 | 4 |
| **Total** | **20** | **20** | **20** | **20** | **20** | **17** | **16** | **18** |

## *5.3 Cost and Power Evaluation - Scale Up Case*

In the scale up experiment, the high-end and low-end server configurations were specified for several different sizes of physical memory. Eight servers were selected and cost and power statistics were gathered for three cases: 1) The server had more than enough memory to allocate as much memory as was needed by each host VM (60-80 VMs in the high-end case and 15-20 VMs in the low-end case). This was referred to as the basic, overprovisioned case, labeled as 100% in the graphs. For the Proliant DL785, the overprovisioned amount of memory for our selected workloads was 64 GB, while for the DL165 the overprovisioned case was 16 GB. 2) Each server was allocated 50% as much memory, and the same workloads were used with memory allocations to each VM that matched test points from the page table simulations. For instance, a workload that would require 502 MB in the overprovisioned case would receive 256 MB in this test case. This allocation is slightly larger than 50% but represents the closest data point obtained from the page table simulations. 3) Each server was allocated 25% as much memory, and the same workloads were run.

The use of DPGAS with remote memory was simulated by doing reallocation of free DRAM memory in each case to simulate sharing of remote memory via one of the suggested memory allocation algorithms that were discussed in Section 3.4.1 and illustrated in Table 4, percentage improvement. The performance statistics from the page table were used to analyze the effects of sharing remote memory; these statistics are compared with the normal case, where memory cannot be shared beyond server boundaries, in Figures 12 and 15. The number of page faults for the overprovisioned case shows no difference between using the normal method and the DPGAS method. This is because the number of page faults only drops when the entire application cannot fit into DRAM, and in the overprovisioned case each application can comfortably fit into DRAM.

### 5.3.1 DL165 - Low-End Case

For the low-end case, eight servers with either 16 (100%), 8 (50%), or 4 (25%) GB were used with VM workload combinations of each of the five benchmarks. Each server was configured with between 15 and 20 VMs, each with a variable number of benchmarks as shown in Table 5. Each instance of a benchmark was allocated the same amount of memory in the normal allocation and DPGAS case. DPGAS allocation could also be used to allocate leftover memory on one server to other benchmark instances on the same server or remote servers. Figure 12 shows the page fault rate for each physical memory size. The page fault rate for most of the tested scenarios was constant except for two exceptions: 1) The 50% scenario using DPGAS had more page faults than the normal memory allocation case because some servers had 4 GB instead of 8 GB, generating cost and power savings but also increasing the overall page fault rate by 14%; and 2) the 25% memory scenario had 3% fewer page faults, due to being able to allocate leftover memory on several servers to benchmarks on other servers that would increase overall system performance.

Figures 13 and 14 show the power and cost reductions achieved by using DPGAS for each initial memory allocation of 16 GB, 8 GB, and 4 GB. In the overprovisioned case, the memory that would not normally be shared across a server boundary can be reallocated by DPGAS, and the overall size of DIMMs in some of the eight servers can be reduced from 16 GB to 12 GB. This produces a cost reduction of 30% (about $2,400) in the overprovisioned case and 27% and 10% in the 8 GB and 4 GB cases. Additionally, due to reductions in DIMM sizes, power is also reduced with DPGAS, from 29% in the 16 GB case to 25% in the 8 GB case. No power reductions were seen for the 2 GB case, because reducing server DIMM sizes from 4 GB to 2 GB did not produce any additional power savings due to low power DIMMs being used for all of the low-end server experiments. The sources used for these DIMMs indicated that normal and low-power DIMMs were both the same price but

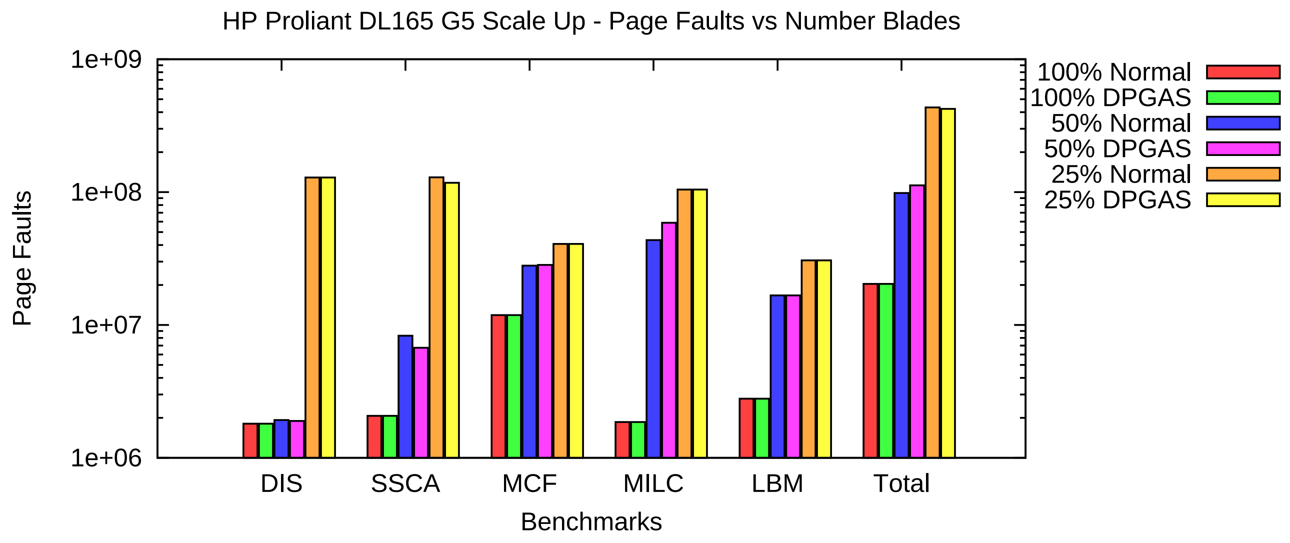


**Figure 12:** Scale Up Performance for Proliant DL165 G5

had slightly different input power budgets.

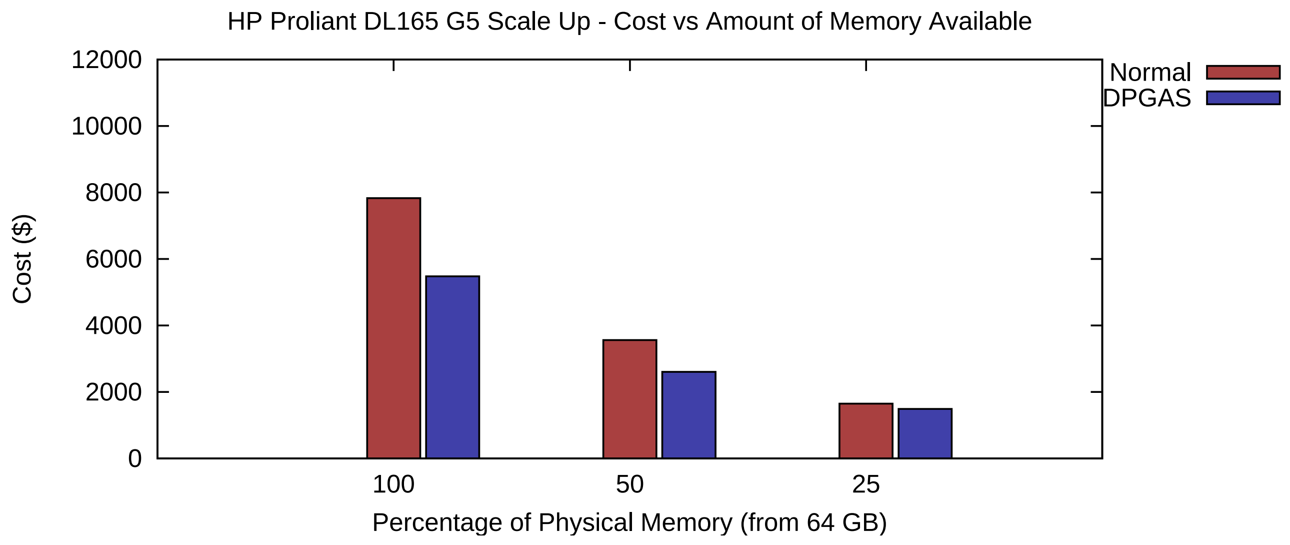


**Figure 13:** Scale Up Cost for Proliant DL165 G5

### 5.3.2 DL785 - High-End Case

For the high-end experiment, eight servers with either 64 (100%), 32 (50%), or 16 (25%) GB were used with the aforementioned VM workloads. In the 50% and 25% cases, servers were not strictly limited to either 32 or 16 GB of memory. In some cases a server was allocated 48 GB or 24 GB in order to improve the performance of the overall system by allowing for more

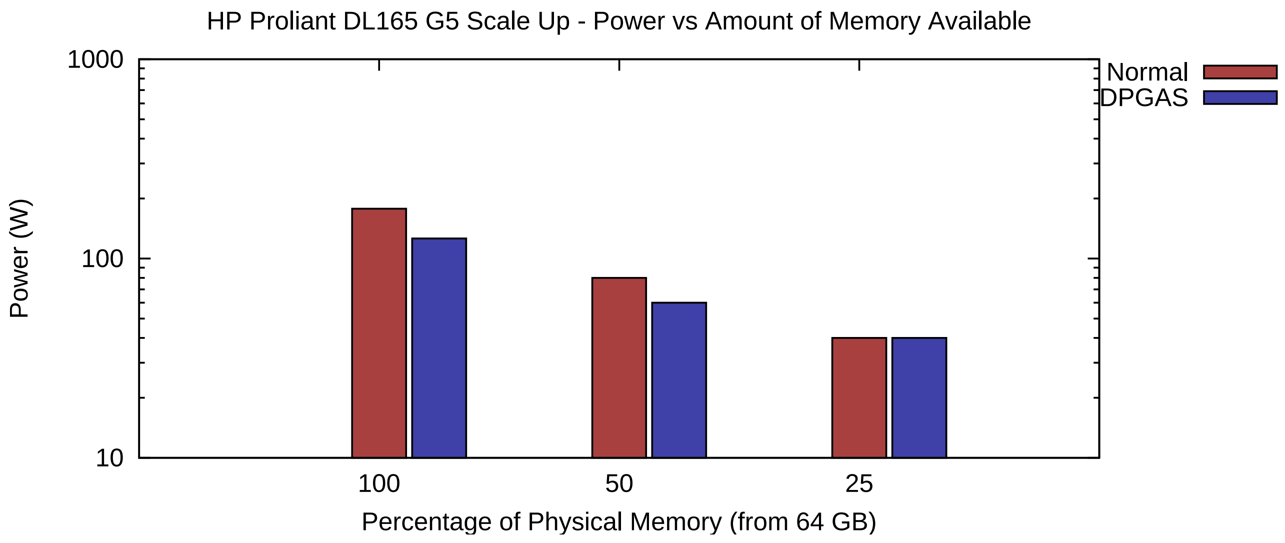


**Figure 14:** Scale Up Power for Proliant DL165 G5

remote memory allocation. In the DPGAS tests, this additional initial memory was usually found to be unneeded, resulting in performance improvements over the normal case, better performance related to the overprovisioned case, and significant power and cost savings.

The performance statistics in Figure 15 show that in the overprovisioned case, DPGAS can be used to reduce the amount of physical memory provisioned across eight servers while reducing the total number of page faults by 59% and 39% for the 32 GB and 16 GB test cases.

Figures 16 and 17 show the cost and power savings for DPGAS-enabled servers. Cost is reduced by 10-19% and power is reduced by 11-18% across the three sizes of memory per server when DPGAS is used rather than normal memory allocation methods.

Thus, for both the low-end and high-end server configurations, the results show that DPGAS can be used to guarantee better cost and power efficiency while also reducing the number of page faults for similarly provisioned servers.

## *5.4 Cost and Power Evaluation - Scale Out Case*

In the scale out case, each server configuration was used to generate statistics for typical large-scale server deployments. A server configuration is usually specified in terms of the number of CPUs that are installed or the number of square feet that racks of servers take

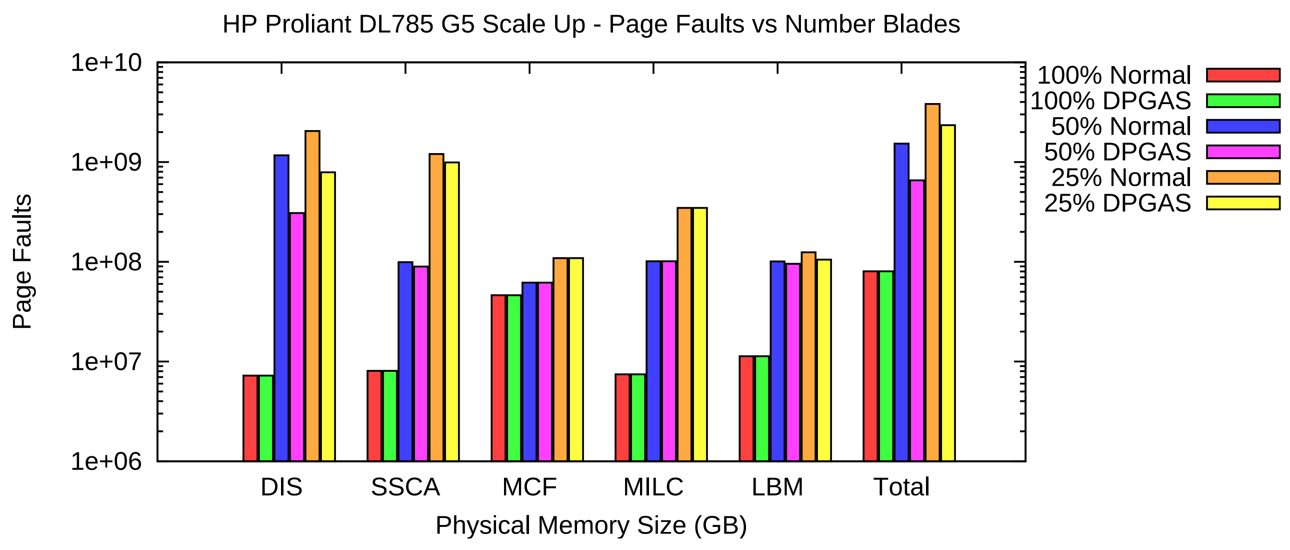


**Figure 15:** Scale Up Performance for Proliant DL785 G5

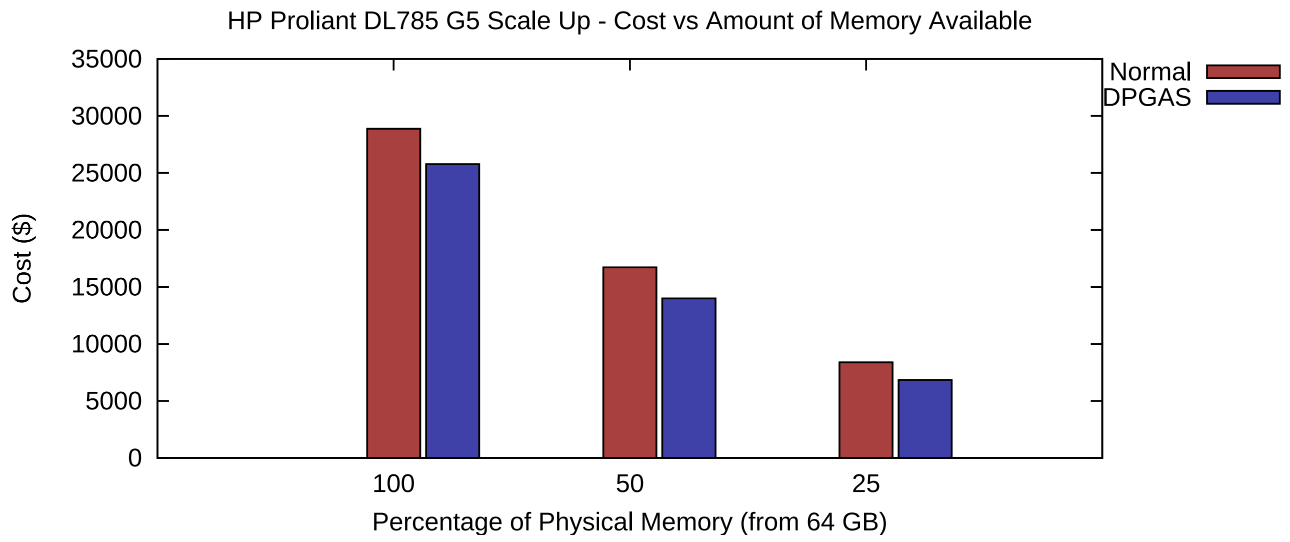


**Figure 16:** Scale Up Cost for Proliant DL785 G5

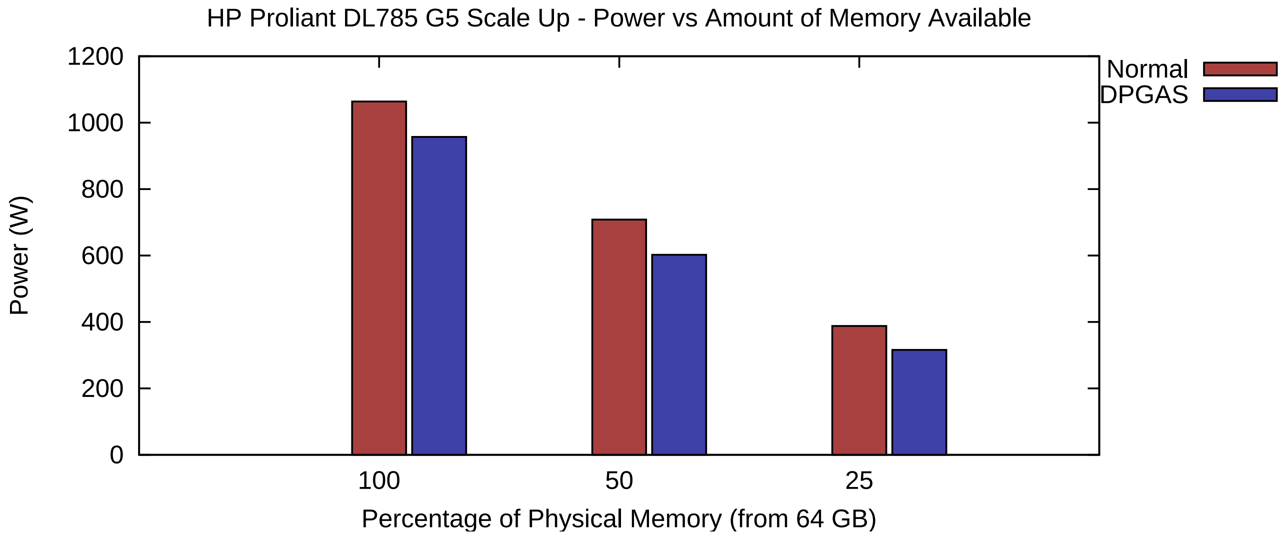


**Figure 17:** Scale Up Power for Proliant DL785 G5

up in a data center—this study chose to specify a configuration in terms of the number of CPUs that were installed. An average number of CPUs for high-performance clusters can range from 1,000 to 4,000 processors [27], while data centers for enterprise applications can run up to tens of thousands of processors. For this study, a base case of 2,000 processors was selected, which fits on 250 high-end servers and can theoretically support about 19,500 virtual machines for the eight-core server. 250 servers were also used for the low-end case, which can support 4,700 VMs across 500 processors. The eight workloads from the scale up case were replicatied across the 250 servers, with six of the workloads being represented on 31 servers each and two of the workloads being represented on 32 servers. To test the hypothesis that DPGAS is useful in reducing cost and power usage while preserving performance, a scale-down of each server install was evaluated using 25 and 50 fewer servers; that is, 225 and 200 servers that supported the same number of virtual machines. By provisioning fewer servers in the data center, not only can memory cost and power be reduced but heating and cooling can also be simplified due to a smaller footprint. Additionally, the reduced memory per server scenario from Section 5.3 was investigated to see how reducing overall memory affects cost, power, and performance.

In order to investigate potential savings with DPGAS, the amount of available memory on each server and free memory on each server was determined. This unused memory can be used to consolidate the overall workload to a smaller number of servers, in essence scaling down the size of the data center. The base case of 250 servers for the high-end server evaluation required a staggering 12.188 TB of DRAM memory with an average of 64 GB of memory for each server, assuming that each server was allocated enough memory for its VMs and additional OS overhead. If each server is allocated 64 GB of memory, then there exists a total capacity of 16 TB at a memory cost of $836,000 and 33,268 Watts just for static and dynamic DRAM power. The large scale of this installation also allows for larger possible savings. By reducing the amount of memory per server to 48 GB from 64 GB, the overall memory cost could be reduced by $245,250 and the overall memory power could be reduced by 9,000 Watts.

### 5.4.1 DL165 - Low-End Case

In the low-end case, performance was investigated for a 500 processor installation and nine different cases: three using 250, 225, and 200 servers each with 16 GB, 8 GB, and 4 GB (or less with DPGAS) of memory. The performance statistics for the entire server installation are shown in Figure 18. As the number of servers is reduced, the number of VMs per server creeps up from 15-20 to 17-23 (225 servers) and 19-25 (200 servers). This increase has two results: 1) The existing memory becomes underprovisioned, which harms performance, and 2) there is less overall unused memory that can be used for remote memory allocations, which reduces cost and power savings for the 50% and 25% initial memory scenarios.

Figures 19 and 20 show the cost and power improvements that DPGAS provides. For the overprovisioned case, DPGAS can be used to reduce the amount of memory on some of the 250 servers, reducing the overall memory costs by 26% or $59,568. When the number of servers is scaled down to 225 servers and 200 servers, cost savings over normal allocation are 15% and 17%. For all three numbers of servers in the 50% initial memory case, cost is reduced by 20% to 27% while the number of page faults is within 15% of the normal allocation setup. The 25% memory experiments show that at a certain point, decreases in cost and power become negligible, about 1% to 5% reduction in cost and no reduction in power (due to factors described in 5.3.1).

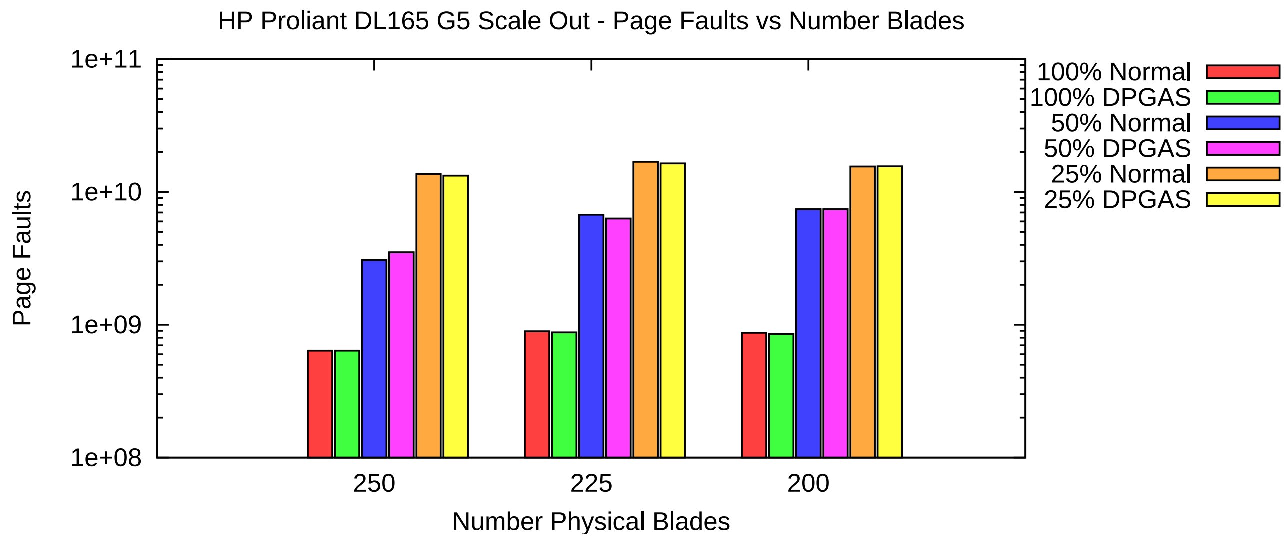


**Figure 18:** Scale Out Performance for Proliant DL165 G5

Power savings range from 14% to 25% for the 100% memory case and 19% to 25% for the 50% memory case, saving 672-1,300 Watts just by using DPGAS instead of normal memory allocation with one of these configurations. If the server memory could be throttled to use only 8 GB per blade with DPGAS, the power consumption from memory would drop from 5,250 Watts to 1,690 Watts, a savings of 3,560 Watts at the cost of 9.88 as many page faults. This same configuration would also save $157,393 in memory costs.

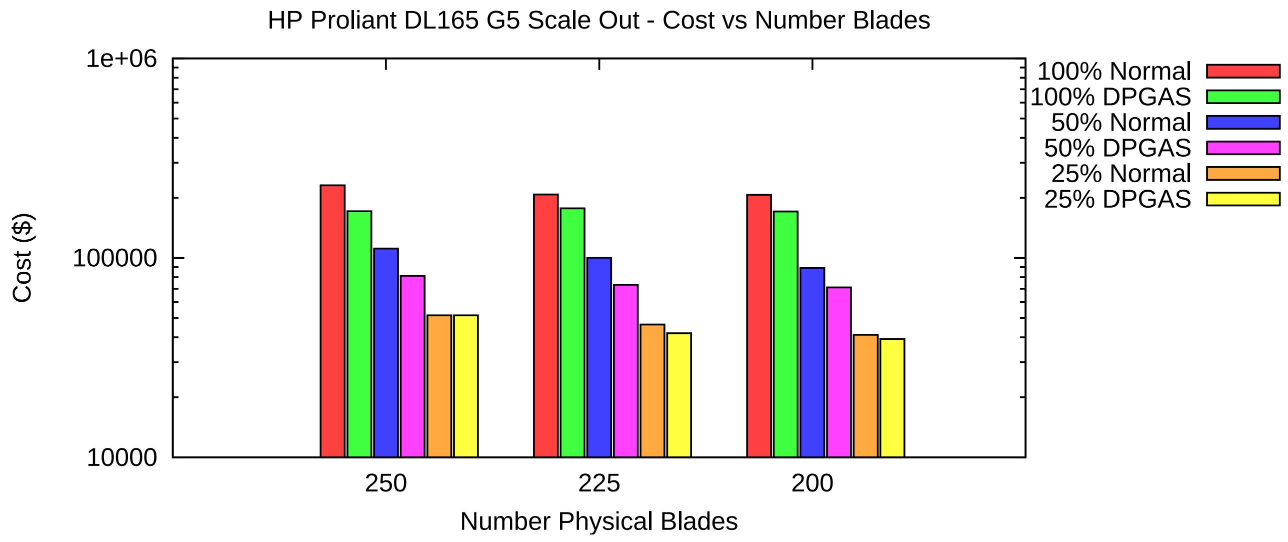


**Figure 19:** Scale Out Cost for Proliant DL165 G5

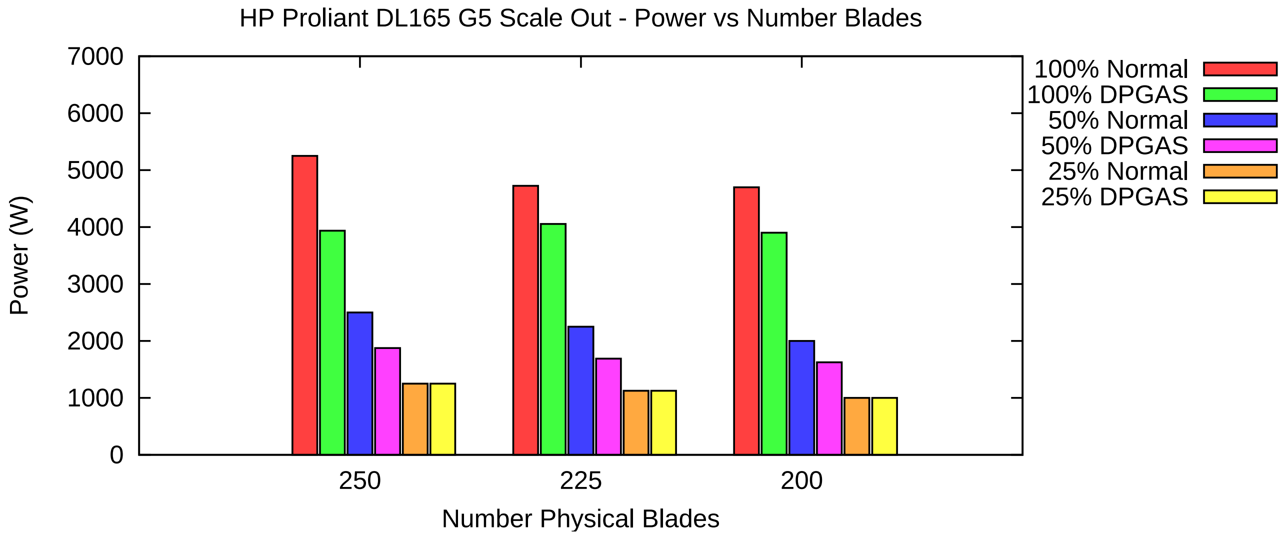


**Figure 20:** Scale Out Power for Proliant DL165 G5

### 5.4.2 DL785 - High-End Case

As in the scale up case, performance was considered to be a secondary constraint to cost and power when deciding how DRAM memory in servers could be initially provisioned. The performance results in Figure 21 show that page faults for the 50% or 32 (or sometimes 48 or 24) GB case were reduced by 57%, 87%, and 29% for the 250, 225, and 200 server scenarios when DPGAS was used as opposed to normal memory allocation. For the 25% case, page faults were reduced by 39%, 51%, and 45% when using DPGAS.

In addition, due to high amounts of unused memory on each server with the high-end servers, the high-end cost and power reductions were much more substantial than in the low-end case. Figure 22 shows that cost was reduced from 14% to 18% for the 250 server scenario, 1% to 7% for the 225 server scenario, and 8% to 14% for the 200 server scenario when DPGAS was used. This translated into a cost savings from \$36,500 to \$154,000. Power was reduced from 14% to 17% for the 250 server scenario, 7% to 9% for the 225 server scenario, and 8% to 13% for the 200 server scenario when DPGAS was used. These results are shown in Figure 23.

Also, if 250 servers could be consolidated onto 200 servers using DPGAS, cost savings could exceed \$619,000 and power savings could exceed 23,000 Watts. However, the number

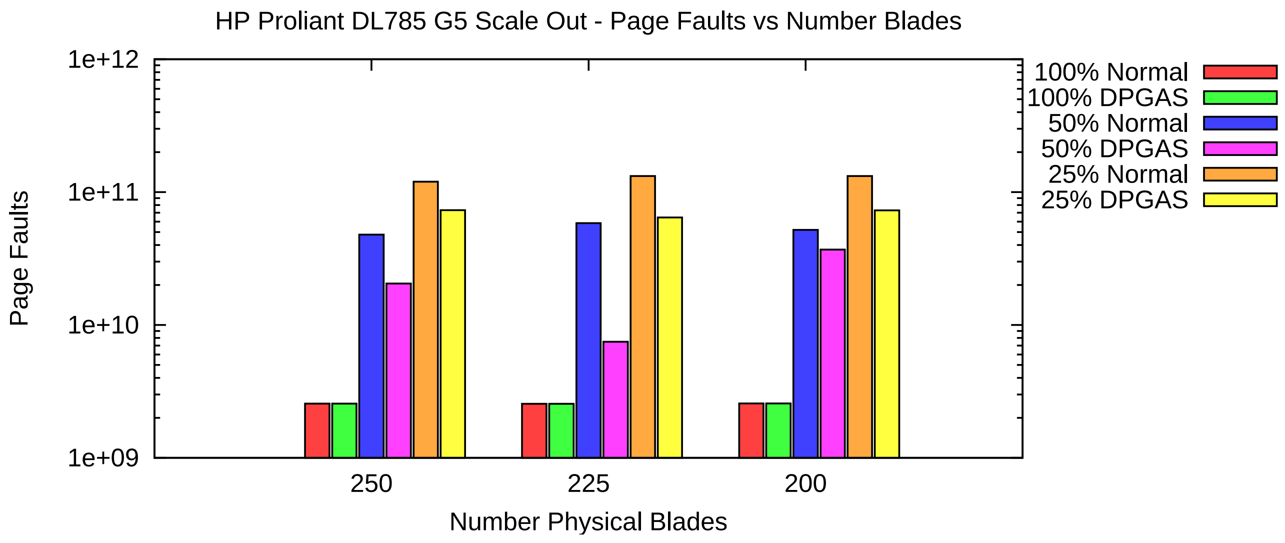


**Figure 21:** Scale Out Performance for Proliant DL785 G5

of page faults would also increase by almost a factor of 30, so this extreme trade-off would not likely be worth the extra savings. Most likely a moderate approach of using 225 servers with 32 GB of memory would be better, resulting in a cost savings of $353,875 and a power savings of 13,338 Watts for only 2.9 times as many page faults as 250 servers using normal memory allocation.

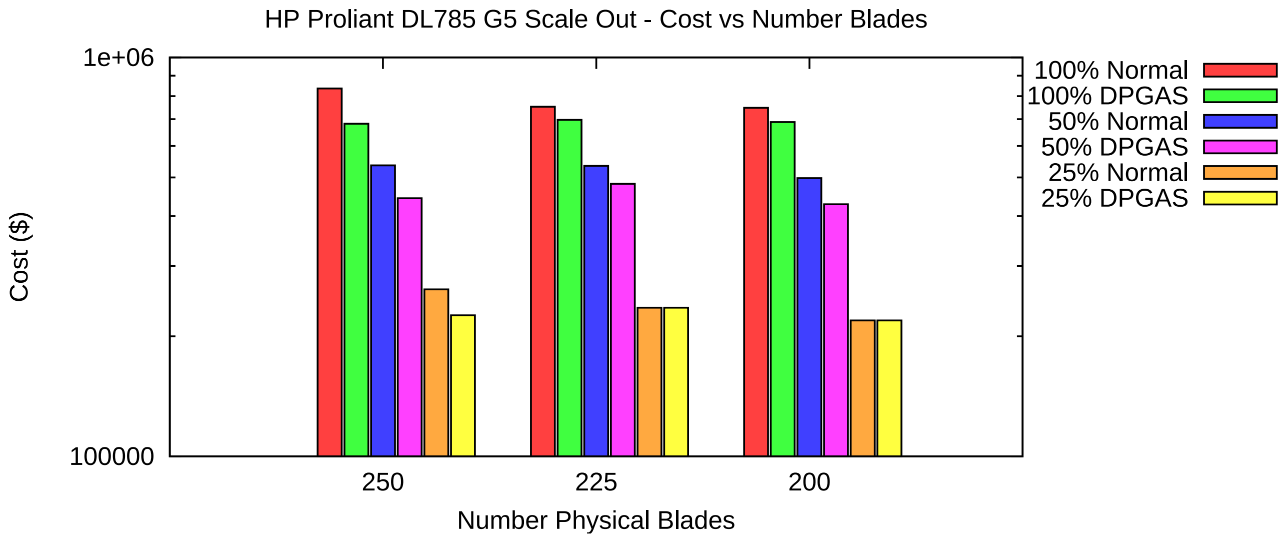


**Figure 22:** Scale Out Cost for Proliant DL785 G5

The cost and power savings in the scale out case is likely much more dramatic than

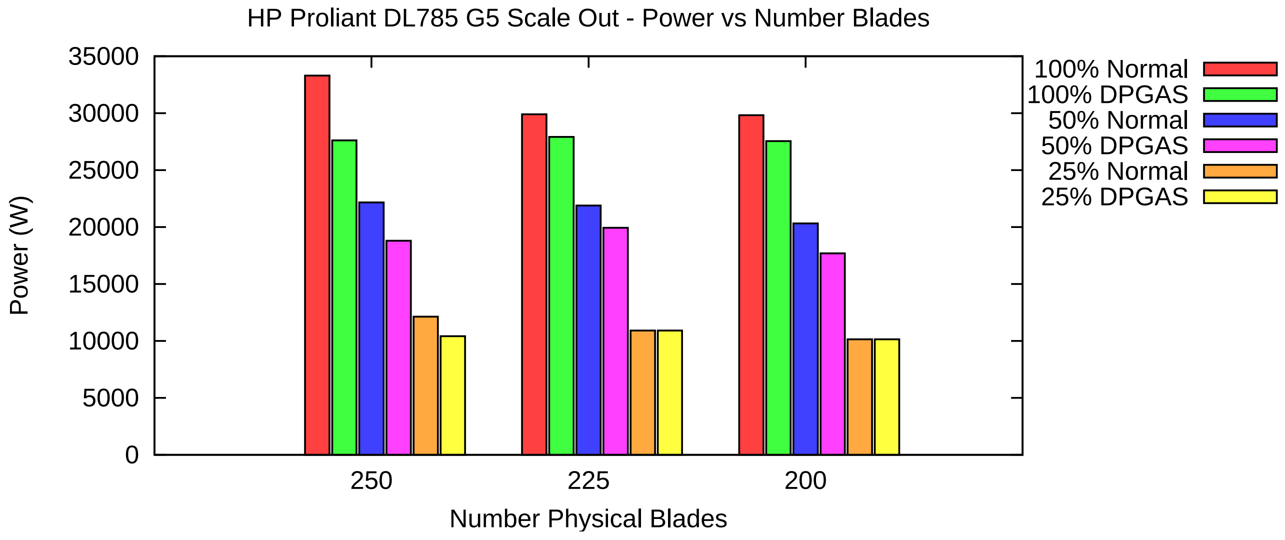


**Figure 23:** Scale Out Power for Proliant DL785 G5

even our results show. By preventing server farms from growing exponentially larger and focusing more on efficiency, we can increase performance/Watt as first discussed by [36]. Page faults are an indicator of performance and runtime in general, but future tests hope to incorporate more detailed timing statistics to better predict DPGAS's effects on runtime and performance/Watt.

## *5.5 DL165 - Performance-Based Analysis*

### 5.5.1 DL165 Scale Up - Performance-Based Analysis

Although DPGAS is being proposed mainly as a technique to improve the cost and power usage of servers, an evaluation was also done for the low-end servers to focus on how performance could be improved if cost and power were kept constant at 8 GB and 4 GB for the 50% and 25% cases. This trade-off resulted in better performance (as shown in Figure 24), but also led to fewer decreases in cost and power as shown in Figures 25 and 26. For the 16 GB case, the usage of DPGAS resulted in a cost savings of 29% and a power savings of 30%, but in the 8 GB and 4 GB tests, the power and cost were the same for the normal and DPGAS allocations.

In the 16 GB case, no performance improvements are seen, but in the 8 GB and 4 GB cases, DPGAS reduces the total number of page faults across eight servers by a factor of

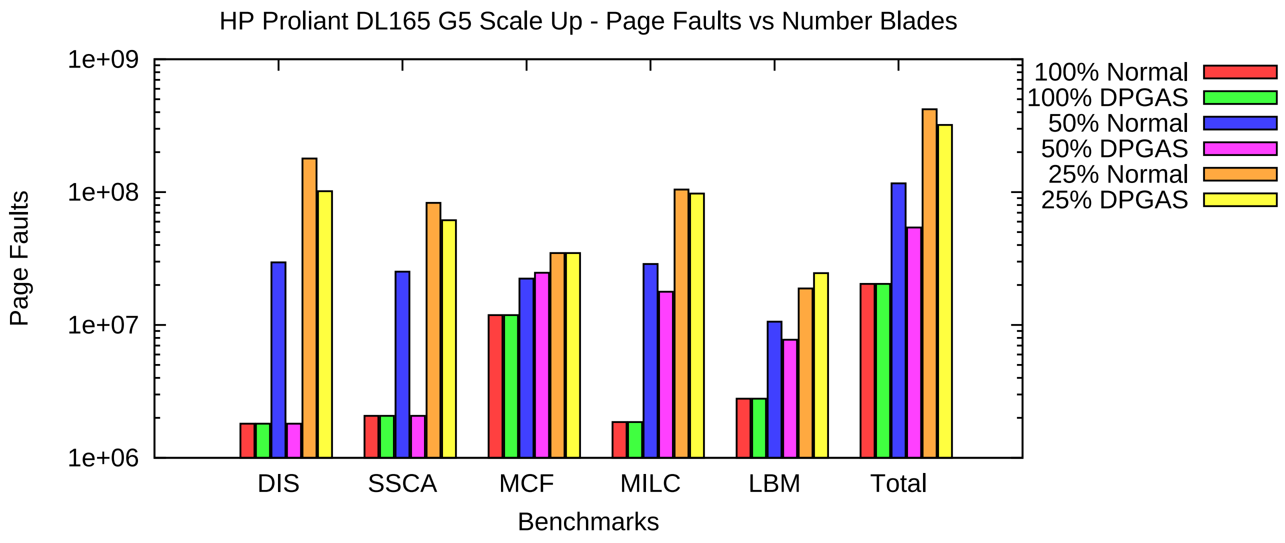


**Figure 24:** Scale Up Performance for Proliant DL165 G5 - Performance-Based DPGAS

53% and 24%, respectively. Additionally, the effects of the DPGAS memory allocation algorithm on individual benchmarks can be seen—the performance of DIS and SSCA increases dramatically (about 94% and 92% for 8 GB and 43% and 26% for 4 GB) compared to the other applications. This is due to the allocation strategy, which used profiling knowledge to determine that these particular applications would most benefit from increases in memory (both through local and remote allocations). Rather than allocating 128 MB of memory to a local instance of MCF, this algorithm asserts that it is in the best interest of system performance to allocate this additional memory to a remote instance of DIS or SSCA.

Compared to the cost- and power-based DPGAS analysis, this experiment had 38% to 52% fewer page faults for the 8 GB case and 16% to 25% fewer page faults for the 4 GB case. These results seem to indicate that in a server that is slightly throttled (closer to the 50% case), DPGAS could also be used with smaller-memory DIMMs and could increase page fault performance to be closer to that of an unthrottled or overprovisioned server. This would keep performance somewhat close to the overprovisioned case while still allowing for memory and power savings by using smaller DIMMs.

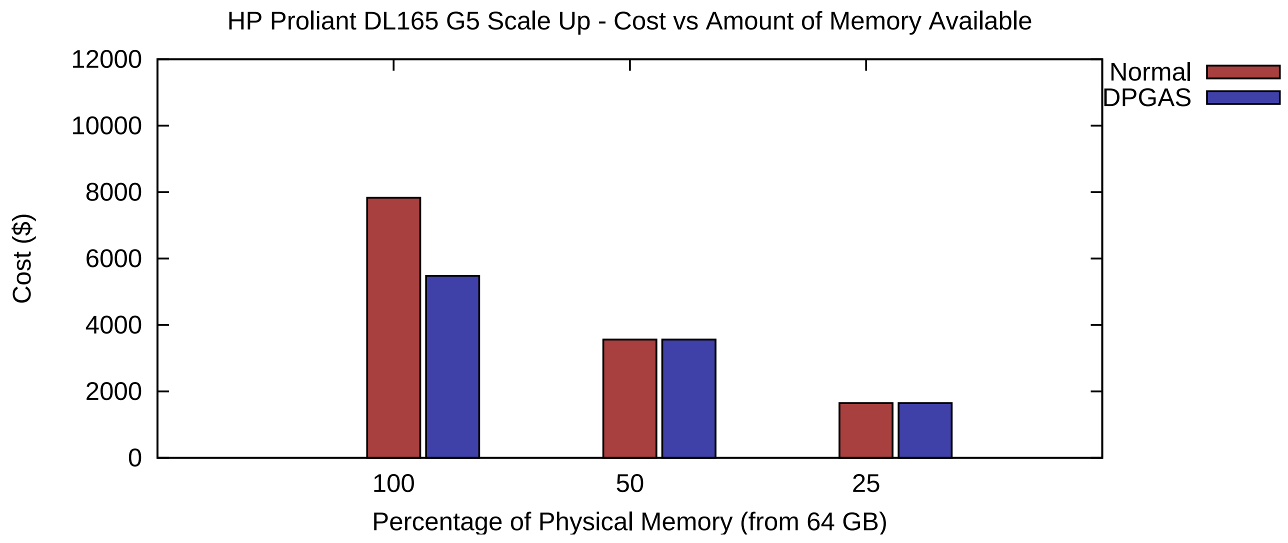


**Figure 25:** Scale Up Cost for Proliant DL165 G5 - Performance-Based DPGAS

### 5.5.2 DL165 Scale Out - Performance-Based Analysis

Figure 27 shows performance results for the low-end server using performance-based DPGAS. In the 250 server case, the usage of DPGAS reduces the number of page faults by 54% and 24% for the evaluations with 8 GB and 4 GB DIMMs in each server. Also, DPGAS increases performance by 34% and 22% for the 8 GB and 4 GB 225 server case and 6% and 9% in the 200 server case for the same memory sizes.

The cost and power results in Figures 28 and 29 show that, with the scale up case, there is not enough overhead to effectively reduce the memory DIMM size, especially since performance was considered to be the first-order constraint. The improvements in cost and power for the DPGAS memory allocation over the normal allocation for the same amount of DRAM ended up being 25%, 14%, and 17% for the 250, 225, and 200 server configurations with 16 GB DIMMs in each server. This translates to a cost savings of $59,568, $30,464, and $35,850 and a power savings of 1,314, 672, and 800 Watts for these respective configurations.

## *5.6 HToE Performance Results*

Xilinx ISE tools were used to synthesize, map, and place and route the HToE Verilog design for a Virtex 4 FX60 FPGA. Synthesis tests using Xilinx software have indicated that the four

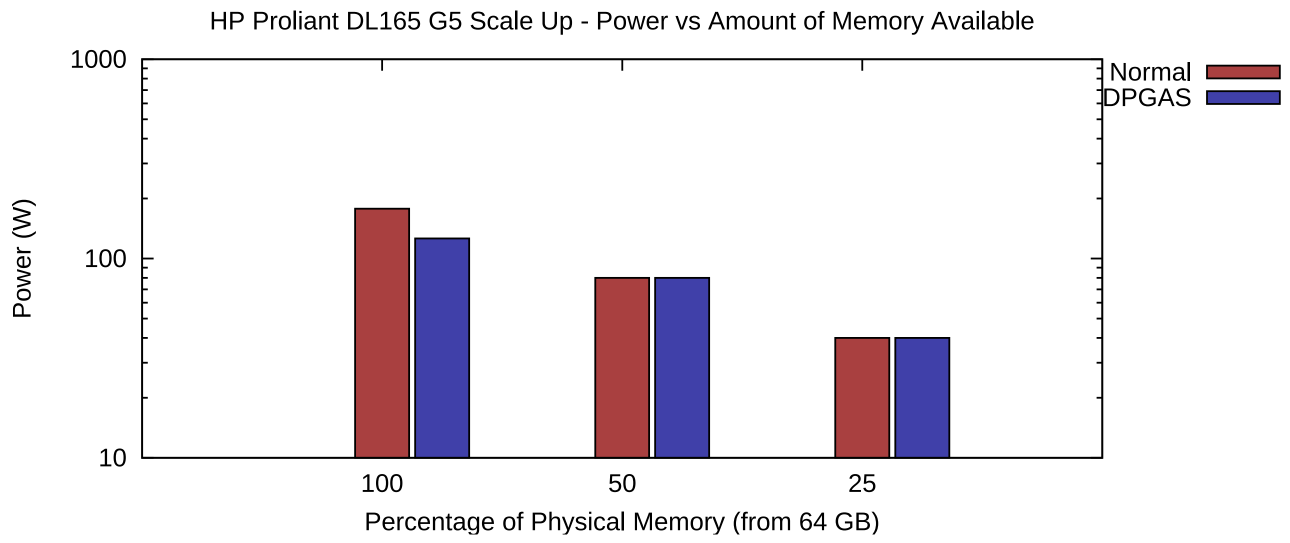


**Figure 26:** Scale Up Power for Proliant DL165 G5 - Performance-Based DPGAS

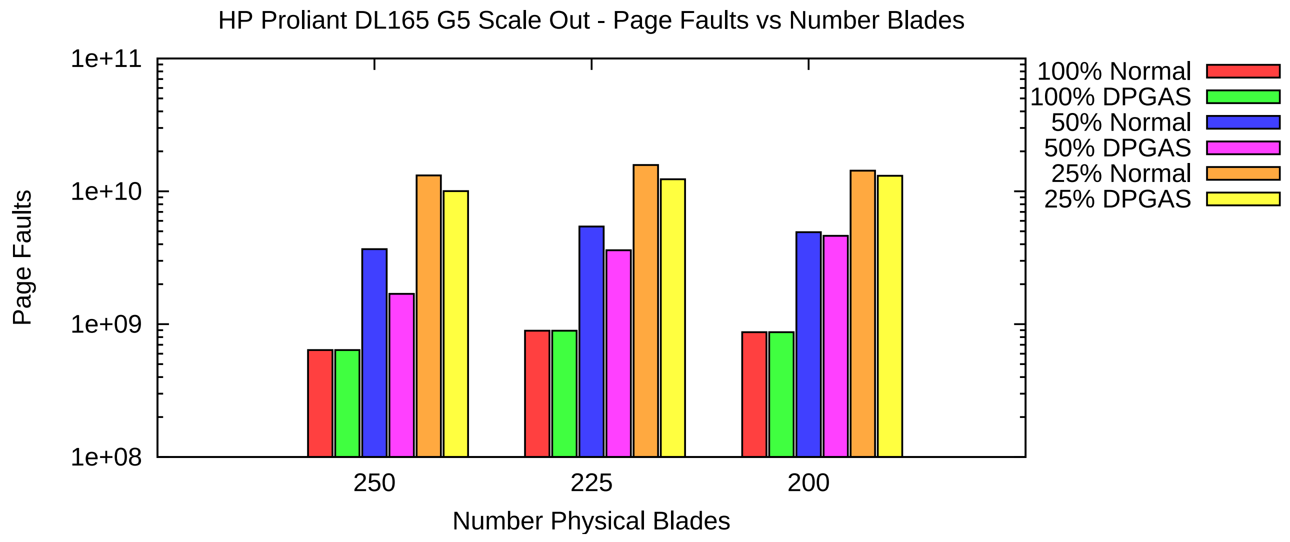


**Figure 27:** Scale Out Performance for Proliant DL165 G5 - Performance-Based DPGAS

modules that make up the bridge are individually capable of speeds in excess of 200 MHz—combined, unoptimized results indicate that the HT bridge is more than capable of feeding a 1 Gbps or faster Ethernet adapter with a maximum speed of 166 MHz. Additionally, estimates using a conservative 125 Mhz (1 Gbps) clock speed and evaluations for each of the request and reply critical paths suggest that the latency overhead of the bridge is on the order of 24-40 ns. In a Xilinx Virtex 4 FX60 FPGA, an unoptimized placement of the bridge uses approximately 1,300-1,500 slices, or approximately 5-6% of the chip. Overheads that

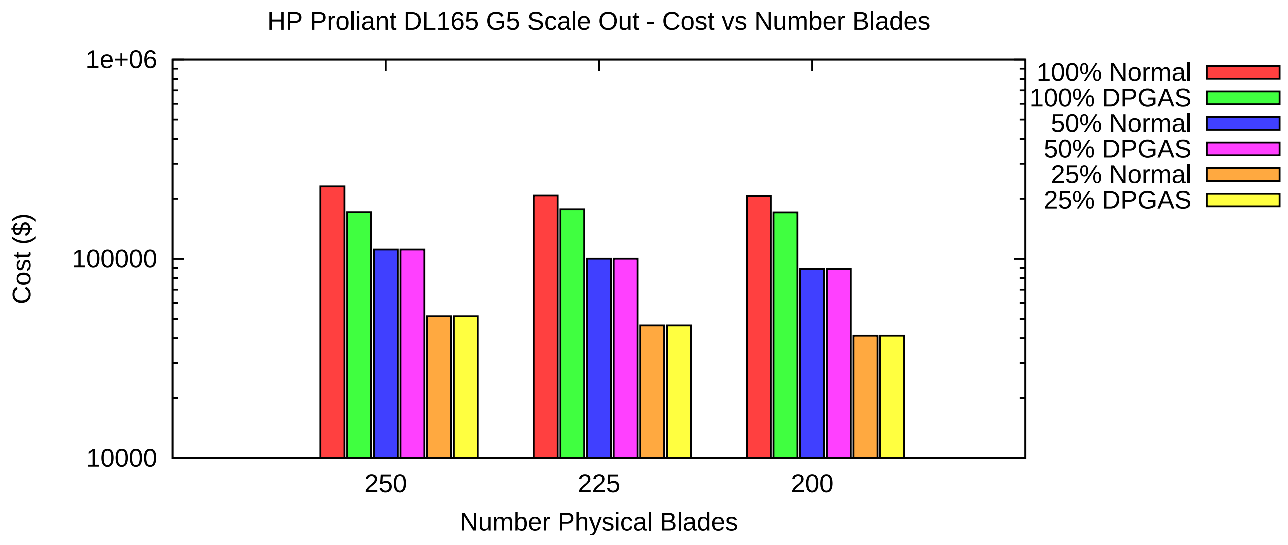


**Figure 28:** Scale Out Cost for Proliant DL165 G5 - Performance-Based DPGAS

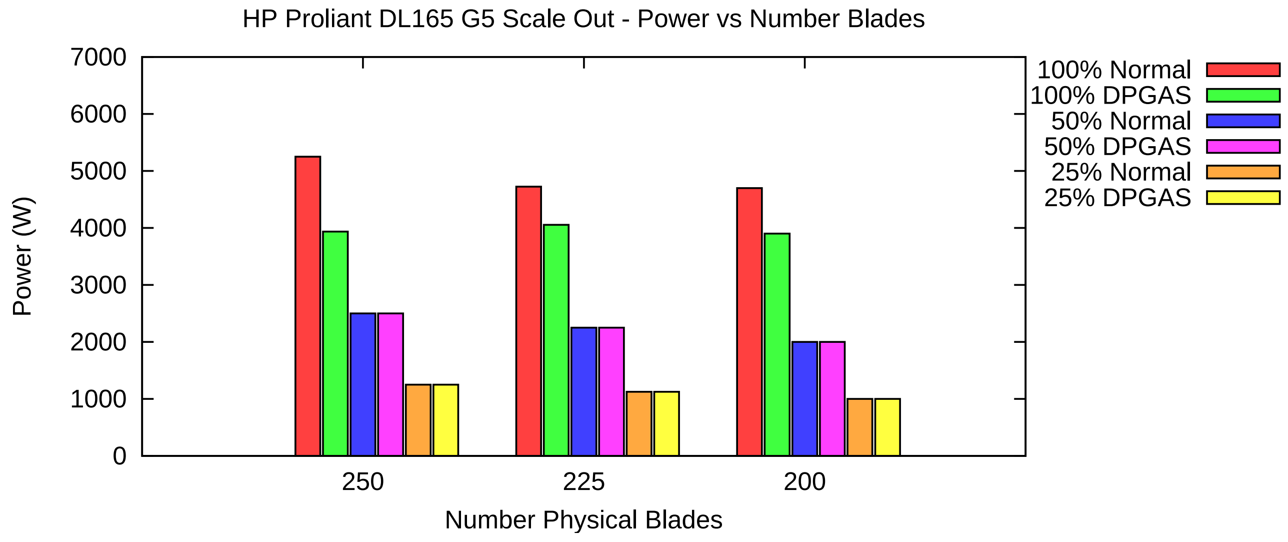


**Figure 29:** Scale Out Power for Proliant DL165 G5 - Performance-Based DPGAS

**Table 6:** Latency Numbers for HToE Bridge

| **DPGAS operation** | **Latency (ns)** |
|---|---|
| Mannheim Core (input) | 55 |
| Mannheim Core (output) | 35 |
| HToE Bridge Read | 40 |
| HToE Bridge Write | 40 |
| Total Read | 1344 |
| Total Write | 712 |

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.

Our results utilize place and routing timing numbers to gauge performance of the DPGAS system because of the lack of suitable FPGAs with Ethernet capabilities. A suitable hardware board would be the Woven card with Xilinx Virtex 4 FPGAs and 10 Gbps Ethernet, but porting the HToE bridge to this platform would also require additional time and expertise due to the higher signaling rate of 10 Gbps Ethernet.

### 5.6.1 Performance Penalties

Table 6 shows the basic performance penalties for using the HToE bridge to access remote memory rather than overprovisioning local memory. The performance penalties are calculated using the formulas:

$$\mathrm{t}_{rem\_req} = \mathrm{t}_{northbridge} + \mathrm{t}_{HToE} + \mathrm{t}_{MAC} + \mathrm{t}_{transmit}$$

where the remote request latency is equal to the time for an AMD northbridge request to DRAM, the DPGAS bridge latency (including the Mannheim core latency), and the Ethernet MAC encapsulation and transmission latency. This general form can be used to determine the latency of a read request:

$$\mathrm{t}_{rem\_read\_req} = 2{*}(\mathrm{t}_{northbridge} + 2{*}\mathrm{t}_{HToE} + \mathrm{t}_{MAC} + \mathrm{t}_{transmit})$$

These latency penalties compare favorably to other technologies, including the 10 Gbps cut-through latency for a switch, which is currently 200 ns [48]; the fastest MPI latency, which is 1.2 $\mu$s [44]; and disk latency, which is on the order of 6-13 ms for hard drives used in the servers studied [53]. Additionally, this unoptimized version of the HToE bridge is more than 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 DPGAS is that workloads are time-varying, and while some applications may perform slightly worse in the short-term, overall power and cost savings are likely to be dramatic.

## *5.7 Further Discussion*

The low-end server demonstrated that while DPGAS is useful in reducing cost and power for cases when server memory is slightly throttled (as in the 16 GB and 8 GB memory cases), it does require enough leftover memory to allocate remote memory to other servers. This scenario illustrated a worst-case workload across eight servers when each workload reached its maximum memory footprint at the exact same time. Realistically, this is unlikely to happen since many workloads go through phases that require different amounts of memory, and each workload most likely is started at a different time or has differing loads based on time of day (as in the case of web servers).

The high-end server configuration demonstrated both performance improvements with DPGAS and cost and power savings due to less strict requirements on how much memory each server could have in the 50% and 25% memory cases. This scenario seemed to show that by making a compromise between performance and cost and power, reasonable improvements in all three areas could be made. In other words, this scenario demonstrated what many system designers already know: performance, cost, and power should all be first-order constraints, if at all possible.

Additionally, the high-end server showed better performance gains than the low-end case simply because there was more free memory to share with remote nodes. Assuming that this was a worst-case evaluation when all workloads were at their maximum memory footprint, the scale out scenario with 250 servers each with 64 GB had about 10,625 GB of unused memory. By scaling the amount of memory on some nodes back to 48 GB, substantial cost and power reductions could be made while keeping page fault rates the same.

The two low-end (cost- and power-biased and performance-biased) server configurations studied illustrated several different traits. The low-end server constrained by a performance-based metric and DPGAS was shown to improve performance but not necessarily power and cost due to the required DRAM to keep the number of page faults closer to the overprovisioned (16 GB) case. Still, reducing the total amount of memory in the system to 8 GB and 4 GB allowed for reasonable cost savings, and DPGAS was used to reduce the memory costs for 250 overprovisioned servers by almost $60,000 total. This is still not an insignificant sum when the base cost for this server was around $1,100, so the cost for these servers without memory would be about $275,000 and DPGAS can save almost 25% of that base cost while maintaining a low number of page faults.

Both of the server configurations used ended up only requiring what could normally be required as the midpoint amount of DRAM. Not all of the benchmarks initially reviewed were selected for a variety of reasons. Some of the trace files gathered illustrated a phase of the benchmark that exhibited very tight spatial locality, meaning that these trace files did not accurately exhibit the memory demands of the benchmark as a whole. Future work will investigate these phases in more detail to make sure that more traces from benchmarks with memory footprints larger than 1 GB can be included in the analysis. Additionally, a very large server workload such as a huge database, search engine, or mapreduce (such as in core databases, Nutch, or Hadoop) would be likely to fully utilize the large amounts of DRAM possible on the high-end server studied while still conforming to a reasonable number of VMs for the number of cores available on each server (meaning that VMs would be memory bound and not computation bound). Further efforts will be made to study the effects of DPGAS on these types of systems as the expected benefits make implementing DPGAS more appealing.

## *5.8 Further Optimizations*

Using DPGAS does not preclude using other power- and cost-saving techniques like dynamic voltage scaling (DVS), load-balancing, and consolidation of VMs. Ideally, combining DPGAS with techniques like VM consolidation could be used to further improve the power/performance and cost/performance of large clusters. Here we discuss some optimizations that have not been tested but are future areas for research.

Our prototype incorporates the usage of Gigabit Ethernet, but DPGAS is also a valid model for other technologies, including Infiniband. Although Infiniband, Myrinet, and Quadrics have exhibited much lower latencies in recent years, the introduction of 40 and 100 Gbps Ethernet is likely to continue to close the latency gap between the technologies due to investments in lower-latency Ethernet switches that work with existing Ethernet clusters and Fibre Channel over Ethernet (FCoE) clusters [16].

Page migration is another possible application to future studies of PGAS in order to more efficiently use memory. By profiling each application and tracking which parts of physical memory are being used most frequently, pages can be migrated from remote nodes to local nodes in order to improve performance for these pages. For instance, the detailed per-page statistics of the page table simulations from this thesis showed that certain virtual addresses are hit at a higher rate, so pages mapping these addresses should be placed in local memory if at all possible. Further evaluation of these trends would be required to substantiate any performance benefits, but it is possible that an idea like the profiling MMC in [4] or profiling using a paravirtualized VM hypervisor (that traps all virtual to physical translations) could be incorporated to provide dynamic performance benefits through page migration.

# CHAPTER VI

# RELATED WORK

Other researchers have also been focused on the growing power and cost implications of large clusters and server farms. Feng, et al [18] 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 [19] with a focus on CPU power. Raganathy, et al [50] 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 [37] cited important reasoning behind why DRAM cost and power should be considered as a major component in improving overall server efficiencies. Additionally, this study proposed techniques for memory compression to reduce overall memory power. Several other researchers have also begun focusing on memory power management at the architecture level, including [31], which proposes using adaptive power-based scheduling in the memory controller, and [17] which uses power "shifting" driven by a global power manager to reduce power of the overall system based on runtime workloads.

At the operating system level, [25] proposed a power-aware paging method that utilizes fast MRAM to provide power and performance benefits. Tolentino [55] [56] also suggested a software-driven mechanism to limit application working sets at the operating system level in order to reduce the need for DRAM overprovisioning.

PGAS has been approached several times in the past five years, mostly as a method for more efficient use of MPI on clusters and as part of the government's high-efficiency computing program [22]. Gasnet [5], X10 [9], and UPC [14] all are projects that are working towards the governments goal for high-efficiency programming languages and supercomputers. DP-GAS is more closely related to Gasnet in that both approaches are focused on low-level or hardware implementation PGAS, although Gasnet is still focused on providing an efficient substrate for writing shared memory and MPI code rather than on increasing system efficiency. Additionally, there are several RDMA-based approaches that have attempted to enable hardware-level remote page swapping, with the most notable attempt being [40].

An evaluation of power and cost trends similar to the ones in this thesis was evaluated in [41], concluding that separate PCI Express-based memory blades could be used to reduce overall memory usage and memory cost and power. [15] investigated real-world statistics for some of the large "warehouse-sized" server farms that Google runs. This study also proposed using dynamic voltage scaling to reduce power usage and utilizing power overheads in these systems to deploy new compute facilities. Additionally, several recent studies have investigated the effects of virtualization on server memory requirements [8] and on how resources are allocated on large clusters and grid-based computers [54].

# CHAPTER VII

# CONCLUSIONS

This thesis proposed a new abstraction called Dynamic Partitioned Global Address Space (DPGAS) for sharing memory between server blades to help reduce the power and cost constraints that are increasingly becoming important with designers of large clusters and datacenters.

A hardware model to facilitate DPGAS called HyperTransport over Ethernet was designed and implemented using Verilog and Xilinx FPGAs to gather timing statistics. These performance statistics were discussed, compared to other current interconnect technologies, and found to be reasonable at 1.3 $\mu$s for an encapsulated HT read operation and 712 ns for an HT write operation.

A page table simulation and analytical model were then used to illustrate the performance, cost, and power effects that memory sharing with DPGAS would have on two real server configurations with eight servers each and two cases of a hypothetical datacenter running real workloads. The two types of scenarios tested, the scale up and scale out scenarios, were picked to show how DPGAS addresses two of the most common ways that datacenter designers manipulate their performance, power, and cooling costs. While it may not be advisable to plan for a datacenter that only has about 25% of the physical memory needed for worst-case workloads, the time-varying nature of workloads along with the usage of DPGAS allows for a significant reduction in the total amount of DRAM required per server blade while preserving performance and supplementing limited on-chip bandwidth.

The analysis of these four server environments showed that DPGAS reduced cost from 10% to 30% for the low-end server scale up experiment and 10% to 19% for the high-end server scale up experiment. Power was not reduced in the low-end server scale up due to small power margins in the 4 GB and 2 GB low-power DIMMs that were used, but DPGAS reduced power by 11% to 18% for the high-end server scale up experiment when compared to normal memory allocation schemes. The data center experiments showed that cost could be reduced by 15% to 26% and 1% to 18% for the low- and high-end scenarios, and power could be reduced from 14% to 25% and 7% to 17% for the low- and high-end scenarios when using DPGAS versus normal memory allocation. Most importantly, DPGAS was shown to be most effective when servers were overprovisioned (had 100% memory allocations), since it helped to smooth out demand peaks by sharing unused memory that would normally remain confined to server blade boundaries. In addition, one set of experiments was run with the low-end server configuration to show that DPGAS could also be used to reduce the overall number of page faults in a system while still keeping memory provisioning constant.

Future extensions to the DPGAS model would focus more on improving memory allocation techniques from the operating systems level and possibly incorporating dynamic profiling information like that offered by Xen's xencontrol management library. More enterprise benchmarks would be useful for future studies including large server workload models like those used in the TCP-C and SpecWeb benchmarks and in real systems that implement web servers like Hadoop and Nutch. Finally, a high-performance port of the HToE bridge to 10 Gbps Ethernet or Infiniband would be a likely candidate for bringing about future adoption of the DPGAS model.

## REFERENCES

[1] Association, I. T., "Infiniband architecture specification volume 1. release 1.2.1," 2008.

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

[3] Bell, C. and Bonachea, D., "A new dma registration strategy for pinning-based high performance networks," in *IPDPS '03: Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, (Washington, DC, USA), p. 198.1, IEEE Computer Society, 2003.

[4] Bellosa, F., "When physical is not real enough," in *EW11: Proceedings of the 11th ACM SIGOPS European Workshop*, (New York, NY, USA), p. 25, ACM, 2004.

[5] Bonachea, D., "Gasnet specification, v1.1," tech. rep., Berkeley, CA, USA, 2002.

[6] Bruening, U., "The htx board: the universal htx test platform."

[7] Burger, D., Goodman, J. R., and Kägi, A., "Limited bandwidth to affect processor design," *IEEE Micro*, vol. 17, no. 6, pp. 55–62, 1997.

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

[9] Charles, P., Grothoff, C., Saraswat, V., Donawa, C., Kielstra, A., Ebcioglu, K., von Praun, C., and Sarkar, V., "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), pp. 519–538, ACM, 2005.

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

[11] "Cray xt3 supercomputer scalable by design," 2008.

[12] "Crucial memory pricing for proliant dl785 g5," 2008.

[13] "Dis stressmark suite updated by uc irvine," 2001.

[14] El-Ghazawi, T. and Smith, L., "Upc: unified parallel c," in *SC '06: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, (New York, NY, USA), p. 27, ACM, 2006.

[15] Fan, X., Weber, W., and Barroso, L. A., "Power provisioning for a warehouse-sized computer," in *ISCA 2007: Proceedings of the 34th Annual International Symposium on Computer Architecture*, (New York, NY, USA), pp. 13–23, ACM, 2007.

[16] "Fibre channel over ethernet in the data center: An introduction," 2007.

[17] Felter, W., Rajamani, K., Keller, T., and Rusu, C., “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), pp. 293–302, ACM, 2005.

[18] Feng, W., “Making a case for efficient supercomputing,” *Queue*, vol. 1, no. 7, pp. 54–64, 2003.

[19] Ge, R., Feng, X., and Cameron, K. W., “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), p. 233.2, IEEE Computer Society, 2005.

[20] Gove, D., “Cpu2006 working set size,” *SIGARCH Comput. Archit. News*, vol. 35, no. 1, pp. 90–96, 2007.

[21] Graham, S. L., Kessler, P. B., and Mckusick, M. K., “Gprof: A call graph execution profiler,” *SIGPLAN Not.*, vol. 17, no. 6, pp. 120–126, 1982.

[22] Graybill, R., “High productivity computing systems.”

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

[24] Henning, J. L., “Spec cpu2006 memory footprint,” *SIGARCH Comput. Archit. News*, vol. 35, no. 1, pp. 84–89, 2007.

[25] Hosogaya, Y., Endo, T., and Matsuoka, S., “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.

[26] “Hp proliant dl servers - cost specifications,” 2008.

[27] “Number of processors share for june 2008 - hpc top 500,” 2008.

[28] “Hp power calculator utility: a tool for estimating power requirements for hp proliant rack-mounted systems,” 2008.

[29] *HP ProLiant DL585 G2 server technology*, 2008.

[30] “Hypertransport specification, 3.00c,” 2007.

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

[32] “Intel 82541er gigabit ethernet controller.”

[33] “International technology roadmap for semiconductors, executive summary 2007,”

[34] Kanter, D., “The common system interface: Intel’s future interconnect,” 2007.

[35] Lameter, C., “Local and remote memory: Memory in a linux/numa system,” 2006.

[36] Laudon, J., "Performance/watt: the new server focus," *SIGARCH Comput. Archit. News*, vol. 33, no. 4, pp. 5–13, 2005.

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

[38] Li, H., Groep, D., and Wolters, L., "Workload characteristics of a multi-cluster supercomputer," in *In Job Scheduling Strategies for Parallel Processing*, pp. 176–193, Springer Verlag, 2004.

[39] Li, H. and Wolters, L., "Towards a better understanding of workload dynamics on data-intensive clusters and grids," *Parallel and Distributed Processing Symposium, International*, vol. 0, p. 60, 2007.

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

[41] Lim, K., Ranganathan, P., Chang, J., Patel, C., Mudge, T., and Reinhardt, S., "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), pp. 315–326, IEEE Computer Society, 2008.

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

[43] "Massif: a heap profiler," 2008.

[44] "Mellanox connectx ib specification sheet," 2008.

[45] "Micron memory datasheets," 2008.

[46] Nethercote, N. and Seward, J., "Valgrind: a framework for heavyweight dynamic binary instrumentation," *SIGPLAN Not.*, vol. 42, no. 6, pp. 89–100, 2007.

[47] Patterson, D. A., "Latency lags bandwith," *Commun. ACM*, vol. 47, no. 10, pp. 71–75, 2004.

[48] "Quadrics qs ten g for hpc interconnect product family," 2008.

[49] "Intel quickpath architecture," 2008.

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

[51] "Rchtx-xv4: High performance computing (hpc) application acceleration board datasheet."

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

[53] "Storagereview.com drive performance resource center," 2008.

[54] Taesombut, N. and Chien, A. A., "Distributed virtual computers (dvc): simplifying the development of high performance grid applications," in *CCGRID '04: Proceedings of the 2004 IEEE International Symposium on Cluster Computing and the Grid*, (Washington, DC, USA), pp. 715–722, IEEE Computer Society, 2004.

[55] Tolentino, M. E. and Cameron, K. W., "Improving the energy efficiency of high-performance server systems," in *SOSP '05: Proceedings of the twentieth ACM symposium on Operating systems principles*, (New York, NY, USA), pp. 1–9, ACM, 2005.

[56] Tolentino, M. E., Turner, J., and Cameron, K. W., "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), pp. 237–246, ACM, 2007.

[57] Wulf, W. and McKee, S., "Hitting the memory wall:implications of the obvious," *Computer Architecture News*, pp. 20–24, 1995.