Final Report

SGER: Dynamic Partitioned Global Address Spaces for Future Large Scale Systems

NSF Grant CCF-0874991

PI: Sudhakar Yalamanchili
School of Electrical and Computer Engineering
Georgia Institute of Technology

1. Introduction

This work explored important facets of the integration of memory systems and interconnection networks. Our point of departure was a proposed model of a global non-coherent address space that is managed in a manner that can lead to significant energy savings in modern and future servers and data centers. Power consumption and cooling power has become the defining attributes of such systems. In the furtherance of this research goal we delved deeper into the memory hierarchy and discovered an important aspect of power management in integrated network and memory systems, namely, innovative use of non-volatile memory. We also investigated what has turned out to be a productive exploration of the use of non-volatile memory technology in the cache hierarchy where the last level cache would be a node on the network. The following sections contain a description of the progress in each of these areas.

2. Non-Coherent Global Address Space Systems

The current solution to satisfying increasing demand for memory on a blade server is to provision memory on each blade for the worst case demand. One recent study empirically measured memory footprints from non-virtualized applications across 3,000 servers under normal applications and found the average physical memory usage to be about 1 Gigabyte [1]. However, this study also found that memory requirements can vary greatly, with 50% of the applications requiring between 1 GB and 4 GB of memory at certain points during the five-week period of data collection. Thus, provisioning blade memory for the average case can prove to be inadequate with respect to the subsequent page fault rate while provisioning for the worst-case memory footprint can lead to servers that are substantially over-provisioned and consequently expensive and power inefficient. Furthermore, the cost of DRAM is a non-linear function of density, speed, and memory size thus small increases in provisioned memory leads to disproportionate increase in cost.

We hypothesize that while memory demands of individual applications can vary substantially, rarely, if ever, do all applications make peak demands concurrently. Further, in the unlikely event that they do a performance penalty in the form of higher page fault rates is acceptable. The idea explored by this work is to reduce the cost and power associated with memory by provisioning blades with less than worst-case memory demand and sharing physical memory across blades during periods of localized, high memory demand. Thus, the physical memory accessible to a blade can vary over time, increasing during

periods of peak load by “borrowing” physical memory from an adjacent blade. This idea of shared memory is clearly not new. However, memory sharing via traditional means can exact significant performance penalties due to interconnect and operating system management functions rendering them infeasible in commodity server configurations.

What has changed is the recent introduction of fast interconnects integrated onto the multi-core die close to the memory controllers. The advent of HyperTransport technology reduced the distance from the “wire” to the on-chip memory controller providing low-latency access to remote memory controllers. Thus, the hardware cost to access remote memory on adjacent blades is no longer prohibitive. However, to productively harness this raw capability a global system model must be defined to direct how the system-wide memory is allocated/accessed and thereby shared across the operating system domains of distinct blades. This is where our approach differs from prior non-uniform memory access (NUMA) architectures. Each blade is under the control of a distinct OS and does not require any specialized support for NUMA hardware, but is likely to have cost models associated with regions of the accessible physical address space. However a blade may periodically become a NUMA machine that has access to a portion of the physical memory of an adjacent blade. The advent of on-die integrated HT makes management of accesses to remote memory feasible from a performance perspective although the appropriate level of granularity remains to be explored. In particular, we expect to be able to adapt known techniques that operating systems have supported in software for page migration and global memory management in clusters.

We have proposed a Dynamic Global Address Space model (DPGAS) by modifying the existing Partitioned Global Address Space model (PGAS) [2] 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. Architectural support for address space management is tightly integrated into the HyperTransport interface to minimize the performance overhead of remote memory accesses and to permit fast, dynamic changes in physical address space mappings. Such a hardware integration has been designed and implemented in synthesizable Verilog using an open source HyperTransport (non-coherent mode) core from the University of Heidelberg. Algorithmically, physical memory is dynamically shared by *spilling* memory demand on a blade to neighboring blades as necessary during peak periods. Consequently, the total amount of memory to be provisioned across the data center can be significantly reduced, leading to substantial cost and power savings with minimal loss of performance (a potential increase in the page fault rate and consequently average memory access times).

Specifically, we have refined the implementation of the DPGAS model and evaluated its potential impact for power savings in a large scale server system. This includes an evaluation of DPGAS with i) traces from memory-intensive applications, ii) an on-demand memory spilling policy to allocate off-blade memory when local demand exceeds available physical memory, and iii) an analytical evaluation of the cost and power savings from more efficient DRAM usage. The major conclusions are reported in a workshop paper [3] and a follow-up conference submission [4]. A summary of the important findings is provided in the remainder of this section.

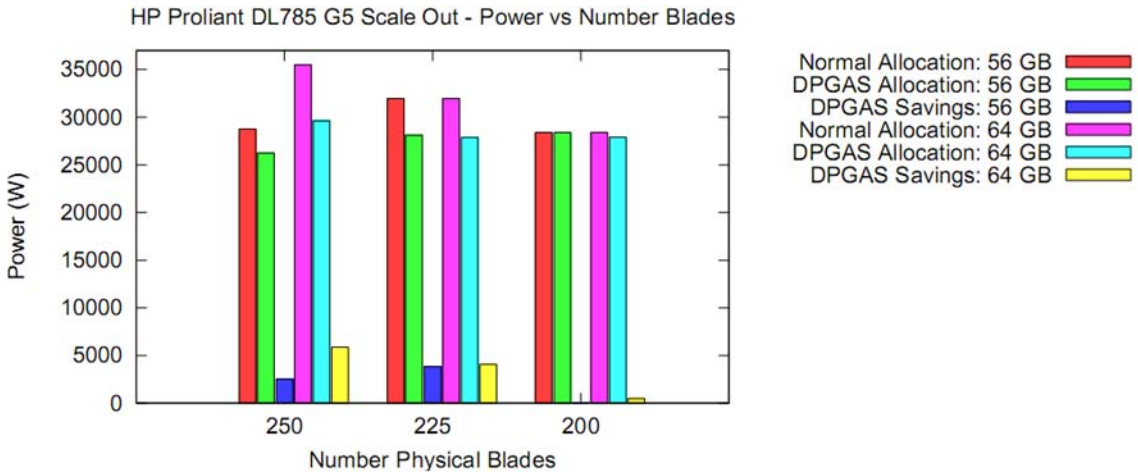

Figure 1 Power Savings with Over-provisioning and Consolidated Applications

2.1 Off-Line Analysis

Our first study utilized an analytic model of workload composition wherein applications with different memory footprints were composed to create a workload memory footprint. The footprints were derived from executions of benchmarks and published memory footprints. Trace based simulation was employed to derive the page fault rates of these applications as a function of memory size by sampling the trace at the point where memory usage was highest. Using a modified bin packing algorithm with an arrival model, we analyzed the *potential* for memory savings that could be translated to power savings. We found that a DPGAS model has the potential to save between 4% and 26% of total memory power in a data center with 250 servers. The power models utilized were configuration models published by HP used in providing guidance for configuring their data centers. We observed that by more efficiently sharing memory, we can significantly reduce over-provisioning that is typical with time-varying workloads in commercial data centers.

In addition to demonstrating the feasibility of DPGAS for saving power, our evaluation also showed the importance of decreasing memory fragmentation to reduce data center power consumption. We found that memory fragmentation can result from many sources, including i) over-provisioning of memory to accommodate peak workloads ii) highly variable memory footprints that increase the difficulty of bin packing applications onto a server iii) physical and logical constraints on servers such as whether applications can be consolidated onto fewer servers or migrated across VM domains.

Figure 1 demonstrates the effects of DPGAS on an over-provisioned (each server is provisioned with 64 GB versus 56 GB necessary as determined statically for a sample workload) and consolidated (mapping a static number of applications on 250, 225, and 200) blades. DPGAS is most effective when memory fragmentation is highest; this situation occurs when server blades are over-provisioned in terms of memory and consolidation is low (64 GB and 250 servers), as is the case in many data centers that have large, time-varying spikes in demand. Figure 2 shows the effect of DPGAS on servers that contain a static number of workloads with either i) similarly sized, large memory footprints or ii) highly variable (both small and large) memory footprints. DPGAS can save 6% to 7% memory power for the first case and 8% to 12% power for the second case. The highly variable memory footprints increase memory fragmentation on each server, which in turn makes DPGAS more appealing for reducing

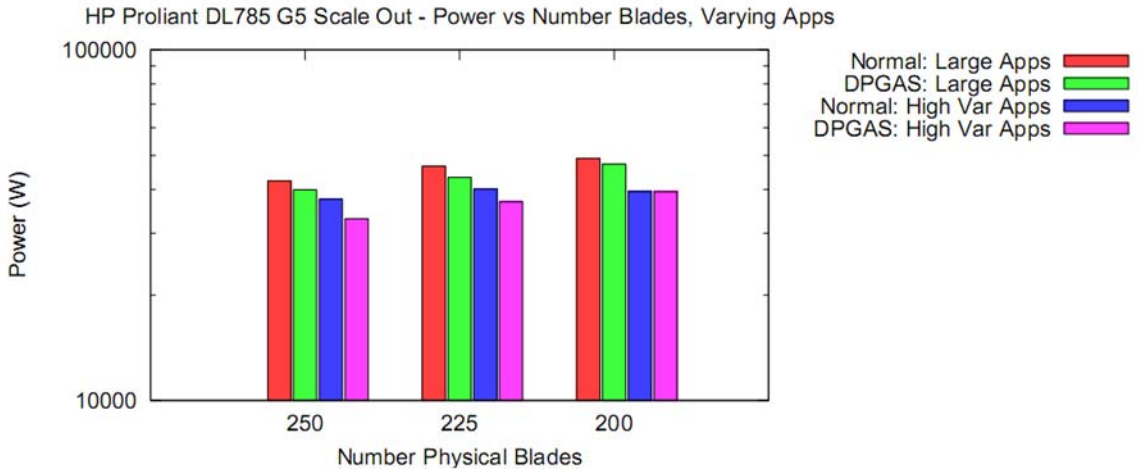

Figure 2 Power Savings with Varying Workload Sizes and Consolidated Applications

memory power via efficient sharing. In contrast, when application consolidation reduces fragmentation, DPGAS-based sharing provides smaller power savings.

2.2 On-Line Simulation-Based Analysis

An implementatation of DPGAS will require fast hardware-based address translation that can be modified at runtime to reflect new allocations of DIMMS across blades. We have designed and implemented (in synthesizable Verilog) such a memory bridge. It is integrated into open source HyperTransport interface IP core available from the HyperTransport Center of Excellence at the University of Hiedelberg [described in 3] and provides a bridge to an Ethernet core available from Xilinx. We could demonstrate that the overhead of address translation was relatively low – on the order of 8-12 clock cycles at 125MHz in an unoptimized FPGA implementation. Using delays instantiated from this implementation we have developed a simulation environment to that enables experimentation with multimode DPGAS systems. The requirements for our simulation environment were that it would be scalable, would contain detailed models of memory and networks, and would be extendable to support other research efforts.

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

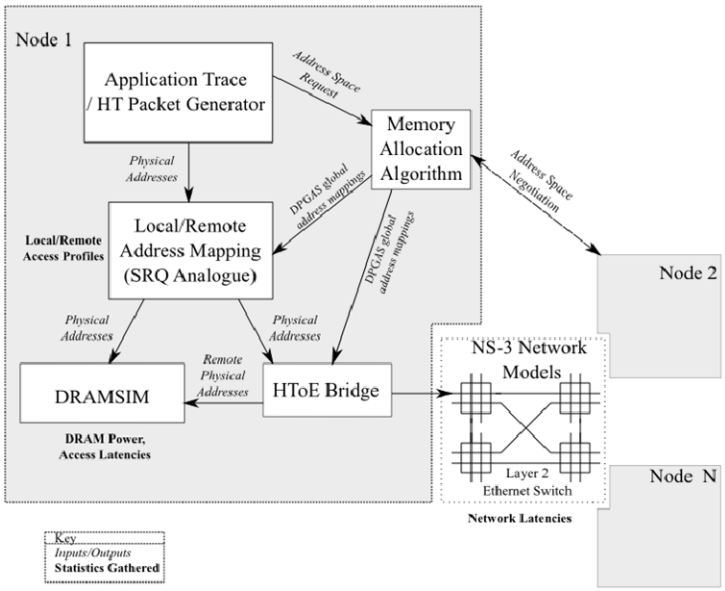

Figure 3 DPGAS Simulation Environment

In addition, our custom code provides a detailed model of the HToE bridge and the Opteron Northbridge's System Request Queue (SRQ) that can be used to map physical addresses to local or remote DRAM [10] [11]. While we are currently focused on trace-driven simulations, we are also planning for integration with future infrastructure such as QEMU to replace traces and improved DRAM models to support newer models of DDR3.

Current studies using this infrastructure include an investigation into the mechanisms of using global address spaces for fair and efficient memory allocation. While other researchers have looked at the usage of low-latency DRAM sharing [12,13] they have not discussed how remote memory allocations can improve power efficiency while preserving fairness in terms of overall performance and sharing between neighboring blades.

The proposed memory allocation algorithms can be implemented at the operating system level using a simple kernel device that communicates via MPI or by standard Unix sockets. The OS will be notified of changes to its available physical memory using the capabilities of libraries like Linux's libnuma or in the case of virtual OS'es, updates to the Virtual Machine Monitor's page tables. The memory allocation daemon will negotiate for changes to the global physical address space with daemons on other nodes and then provide dynamic mapping updates to the System Request Queue using Function 1 Address Map Registers [10] and write packets to the HToE bridge. Updates to the underlying hardware and operating system will permit transparent remapping of remote memory accesses without requiring application involvement. Since our current infrastructure uses application traces instead of an actual operating system, we have labeled this daemon as the memory allocation algorithm in the figure above.

We have proposed two different memory allocation techniques to share remote DRAM at a cache-line (64 Byte) granularity - the *spill/receive* model and a *credit-based* model. The spill/receive model designates certain nodes as "spill" nodes that send some of their data to remote "receivers", allowing for novel opportunities in provisioning remote memory for irregular topologies and for creating "fat" or over-provisioned nodes and "thin" or minimally provisioned nodes. A variation of this scheme that we plan to test includes the concept of remote memory blades as described in [13]. Memory blades can make multiple requests to share remote memory on "receive" nodes, but overall fairness is directed by available memory and a least recently requested mechanism to break request deadlock and release remote memory to another "spill" node.

Credit-based memory allocation specifies that nodes must have credits to request remote memory, and credits are distributed equally or according to application workload during global epochs of time. If a node runs out of credits, it must swap out pages to disk that potentially correspond to live cache lines and then suffer additional performance penalties. For both memory allocation techniques, nodes are encouraged to share remote memory (and negotiate for credits) with their nearest neighbors to reduce hop count.

Future studies that will be made possible by this infrastructure include network topology studies, flow control and Quality of Service studies for HToE, and studies involving the usage of DPGAS for sharing application data across nodes with multiple readers and writers (consistency model support).

3. Novel Uses of Non-Volatile Memory in the Cache Hierarchy

To further manage power consumption in such systems we started to address the use of non volatile memory as a replacement to some of the high energy consumption components such as caches in conventional processor architectures. In particular we were interested in using non-volatile memory at a suitable place in the memory hierarchy rather than as just another level in the memory hierarchy between DRAM and disk. Concurrently, in related research we discovered power supply gating of cores to be a valuable mechanism for effectively controlling power consumption in many core architectures. However, the main difficulty from a performance point of view was that core state and the cache state had to be loaded after each power cycle. Consequently we started to examine the same non-volatile memory technologies higher up in the memory hierarchy since caches, DRAM controllers, and high performance network interfaces such as HyperTransport, QPI and PCI Express are becoming logically and architecturally more tightly integrated as we documented in a recent publication [5]. We started our investigation with Spin Torque Transfer RAM (STTRAM) which appeared to offer the best combination of density, access time, and power attributes. The preliminary results of these studies in collaboration with the circuits group of Professor Mukhopadhyay at Georgia Tech are documented in the *International Conference on Computer Aided Design* in 2009 [6] and a paper in the *IEEE Transactions on VLSI* [7]. Finally, a paper documenting architectural optimizations for STTRAM appears on the *Proceedings of the International Symposium on Low Power Electronics and Design* (ISPLED) [8]. This summary documents the major results that we have contributed to those papers.

A major challenge with STTRAM technology is the high write currents. Consequently, the ratio of read operations to write operations to a cell has a significant impact on the energy consumption for a cell.

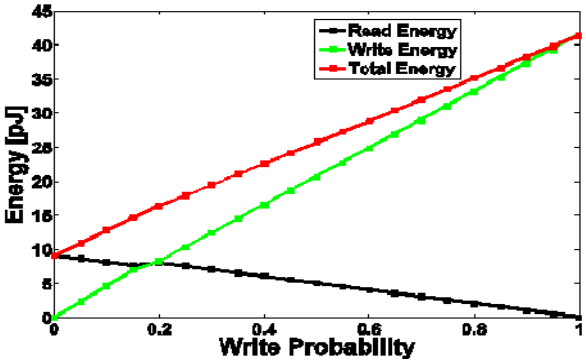

Figure 4 Behavior of Read/Write Energies

Our studies revolved around implementing the L2 cache with STTRAM technology. Our initial efforts studied the read/write statistics of benchmark programs and the read/write energies of cell access (in collaboration with Professor Saibal Mukhopadhyay's group) as delineated below.

STTRAM provides a large area and energy benefit for caches when compared to the conventional SRAM cells, due to the reduced cell structure. For an STTRAM based array a variation of 0.1 in the write probability can change the write energy of the cell by ~8pJ. Note the behavior tracked in Figure 4. This observation is the key motivation for further cache design strategies to minimize the number of writes even at the cost of increasing the number of read accesses since the energy consumption is significantly reduced. Several optimizations can be applied to reduce the number of cell writes. For example, if data was identified as *read-once* data at the application level then such a read operations need not maintain multi-level inclusion reducing the number of write operations to the lower level cache and reducing the ratio of the number of writes to reads in the lower cache levels. Additionally we have discovered and proposed (see below) modified replacement policies, for example, dirty lines are kept longer in the higher levels of the cache and non-dirty lines are evicted earlier.

The initial experiments were conducted using Zesto – an x86 microarchitecture simulator (zesto.cc.gatech.edu). Additionally a set of offline trace analysis tools were developed to quantify the penalties imposed by the above mentioned cache optimizations. Properties such as the number of cycles between accesses to a specific address or the number of reads and writes within a memory stream are metrics that can be used to judge the effectiveness of these optimizations. Early results of the analysis a number of benchmarks to observe the read/write probabilities as a function of the cache size are shown in Figure 5.

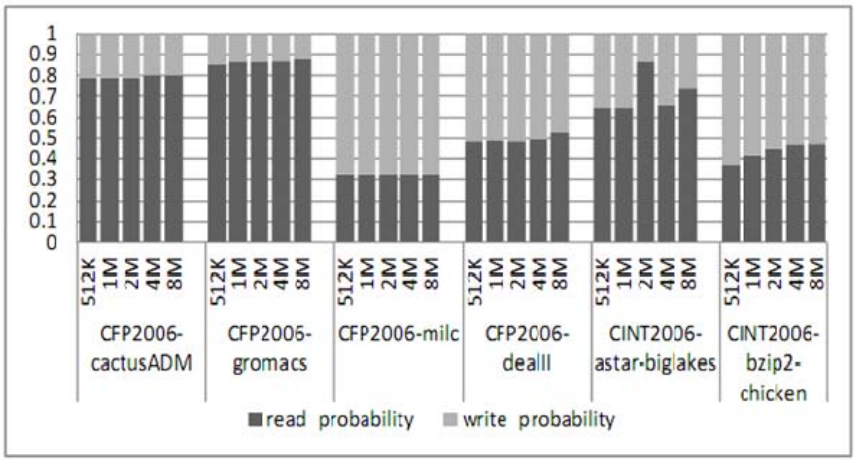

Figure 5 Read/Write Probabilities as a Function of L2 Cache

The project has supported one graduate student researcher – Ms. Mitchelle Rasquinha - who during this project spent approximately 7-8 months as an intern at Intel Corporation working in the area of the "uncore" – interconnection networks and memory systems. Her internship was motivated in part by this research project and she is now positioned well to build on her experiences in the furtherance of her research efforts. We believe the internship experience has been a valuable educational and technical training experience.

Having understood the major limitations in STTRAM we studied architectural techniques to combat the high energy writes and hence improve its use as a universal memory element. Over the past 4 months we studied two techniques namely 1. Write Biasing (WB) 2. Write cache (WC); that can be used to minimize the write energy consumption in a cache array using STTRAM cells.

WB is a cache replacement policy adopted in the level of the cache hierarchy above the cache level implemented with STTRAM arrays. Given a specific cache configuration, and working with our collaborators in the circuit design space we designed both SRAM and STTRAM cells to meet a fixed cycle delay for reads and writes. The write energy for STTRAM per cell was around 260 pJ for a 64x128 memory array which was a 2X increase over its SRAM counterpart. The WB scheme was then applied to the cache level above the STTRAM based cache so as to keep dirty cache lines within that level at the cost of increased evictions for read only lines. When there is a cache line hit for a clear line the general LRU policy would update its status to be most recent. Since reads occur more frequently than writes the writes tend to occupy lower positions of the LRU stack and are hence evicted earlier than clean cache lines. We modified the policy to promote read hits to a position 'K' where 'K' is the distance from the top of the stack. We studied different values of 'K' and found that the optimal value for minimum energy consumption varies for different applications. This is heavily dependent on the ratio of loads to stores in applications and further if the loads were to stores with relatively close inter reference times. Several online techniques may be adapted to tune the parameters for such an optimization. We studied the policy in two configurations. First SRAM DL1 with WB for its replacement policy, coupled with an

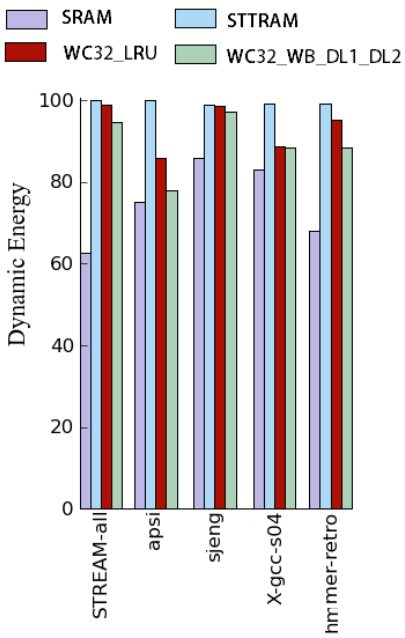

Figure 6 Dynamic energy variation for different configurations

STTRAM-DL2. Second we studied a much larger STTRAM LLC coupled with the WB policy applied to a STTRAM DL2 and an SRAM based DL1. In either case the benefits of reduced energy was seen from careful tuning of the policies for specific memory access patterns.

The WC technique was a second technique adopted to minimize write operations to the STTRAM array. This approach used a second fully associative 32/64 way cache that is accessed in parallel to the L2 and therefore does not affect the latency of the L2 hits. The write cache aims to coalesce relatively higher inter-arrival time (IRT) store requests. Thus any store from the DL1 writes into the write cache instead of the L2. If the line is present in the L2, it is first transferred to the write cache, and then the store is performed. This line transfer is modeled on the same bus that connects the L2, L1 and the Write cache. Considering the large write latency of a write back this transfer of lines does not affect our latency adversely while it increases energy efficiency.

We studied benchmarks from a wide range of benchmark suites with applications that were both compute and memory intensive. Figure 6 shows a subset of those results. As can be seen the aim was to reduce the dynamic energy as far as possible to its SRAM equivalent. We found varying degrees of energy gains from the WB and WC optimizations depending mainly on the ratio of writes to reads and their locality. WB when employed at the higher level cache would coalesce stores with a small IRT and the write cache would further coalesce stores with higher IRT and bypass them to the lower level.

While an STTRAM cell can be designed to meet very short access times almost equal to that of SRAM the technology trends demonstrate how write current increase at low latencies is exponential. One of the approaches to using STTRAM would be to design the cell at increased latencies. However in existing micro architecture designs this would translate to stalling the pipeline which is not desirable. Figure 7 summarizes the design space we studied for varying cache configurations. When used as a substitute to SRAM cells and increasing the cache sizes for a constant area analysis the latency benefits from the larger cache is immediately apparent. However the energy consumption increases by ~40%. A fixed logical size analysis however does not have the

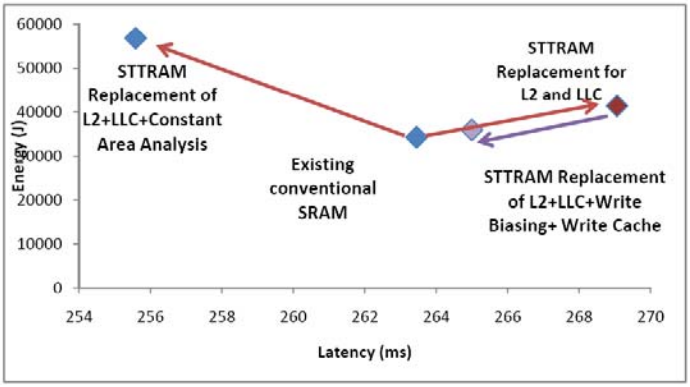

Figure 7: Design Space Exploration

increased energy cost but a performance loss. We them propose optimizations within the micro-architecture that further reduce the performance and energy cost to recover most of this increase.

4. Publications

This research resulted one invited book chapter (reference 5) four refereed conference publications (references 3, 6, 7, and 8) and one refereed conference publication under review (reference 4).

5. References

1. S. Chalal and T. Glasgow. Memory sizing for server virtualization. 2007. http://communities.intel.com/docs/.
2. Philippe Charles, Christian Grothoff, Vijay Saraswat, Christopher Donawa, Allan Kielstra, Kemal Ebcioglu, Christoph von Praun, and Vivek Sarkar. X10: an objectoriented approach to non-uniform cluster computing. In OOPSLA '05, pages 519–538, New York, NY, USA, 2005. ACM.
3. J. Young, S. Yalamanchili, J. Duato, and F. Silla, "A HyperTransport-Enabled Global Memory Model For Improved Memory Efficiency," *First Workshop on HyperTransport Research and Applications*, February 2009.
4. J. Young and S. Yalamanchili, "Dynamic Partitioned Global Address Spaces for Power Efficient DRAM Virtualization," *submitted to First International Conference on Green Computing*, 2010.
5. S. Yalamanchili and J. Young, "System Impact of Integrated Interconnects," (Invited), in *High Performance Communication: A Vertical Approach*, Collection on High Performance, September A. Gavrilovska eds., CRC Press, Taylor & Francis Group, 2009
6. S. Chatterjee, M. Rasquinha, S. Mukhopadhyay, and S. Yalamanchili, "A Methodology for Robust, Energy Efficient Design of Spin-Torque-Transfer RAM Arrays at Scaled Technologies," *Proceedings of ICCAD 2009*.
7. S. Chatterjee, M. Rasquinha, S. Mukhopadhyay, and S. Yalamanchili, "A Scalable Design Methodology for Energy Minimization of STTRAM: A Circuit and Architecture Perspective,", to appear in *IEEE Transactions on VLSI*.
8. M. Rasquinha, D. Choudhary, S. Chatterjee, S. Mukhopadhyay, and S. Yalamanchili, "An Energy Efficient Cache Design Using Spin Torque Transfer RAM (STTRAM), *Proceedings of the International Conference on Low Power Electronics and Design (ISPLED)*, August 2010.
9. Luiz A. Barroso and Urs Hölzle, The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, Morgan and Claypool, 2009
10. BIOS and Kernel Developer's Guide (BKDG) For AMD Family 10h Processors. September 2007. http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/31116.pdf
11. Conway, P. and Hughes, B. The AMD Opteron Northbridge Architecture. *IEEE Micro* Vol. 27, 2, Mar. 2007, pg. 10-21.
12. S. Liang, R. Noronha, and D. K. Panda, "Swapping to remote memory over infiniband: An approach using a high performance network block device." September 2005.

13. Lim, K., Chang, J., Mudge, T., Ranganathan, P., Reinhardt, S. K., and Wenisch, T. F. Disaggregated memory for expansion and sharing in blade servers. *SIGARCH Computer Architecture News* Vol. 37, 3 June 2009, pg. 267-278.