Using System Emulation to Model Next-generation Shared Virtual Memory Clusters

Angelos Bilas, Courtney R. Gibson, Reza Azimi, Rosalia Christodoulopoulou*, and Peter Jamieson

Department of Electrical and Computer Engineering
University of Toronto
Toronto, Ontario M5S 3G4, Canada
{bilas,gibson,azimi,jamieson}@eecg.toronto.edu

*Department of Computer Science
University of Toronto
Toronto, Ontario M5S 3G4, Canada
roza@cs.toronto.edu

Abstract

Recently much effort has been spent on providing a shared address space abstraction on clusters of small–scale symmetric multiprocessors. However, advances in technology will soon make it possible to construct these clusters with larger–scale cc-NUMA nodes, connected with non-coherent networks that offer latencies and bandwidth comparable to interconnection networks used in hardware cache–coherent systems. The shared memory abstraction can be provided on these systems in software across nodes and hardware within nodes.

Recent simulation results have demonstrated that certain features of modern system area networks can be used to greatly reduce shared virtual memory (SVM) overheads [19], [5]. In this work we leverage these results and we use detailed system emulation to investigate building future software shared memory clusters. We use an existing, large–scale hardware cache–coherent system with 64 processors to emulate a complete future cluster. We port our existing infrastructure (communication layer and shared memory protocol) on this system and study the behavior of a set of real applications. We present results for both 32- and 64-processor system configurations.

We find that: (i) System emulation is invaluable in quantifying potential benefits from changes in the technology of commodity components. More importantly, it reveals potential problems in future systems that are easily overlooked in simulation studies. Thus, system emulation should be used along with other modeling techniques (e.g. simulation, implementation) to investigate future trends. (ii) Our work shows that current SVM protocols can only partially take advantage of faster interconnects and wider nodes due to operating system and architectural implications. We quantify the related issues and identify the areas where more research is required for future SVM clusters.

I. Introduction

Recently, there has been a lot of work on providing a software a shared address space abstraction on clusters of commodity workstations interconnected with low-latency, high-bandwidth system area networks (SANs) [12], [18], [19], [29], [3], [9], [25], [24], [27], [4], [15]. The motivation for these efforts is two-fold: (i) System area network (SAN) clusters have a number of appealing features: They follow technology curves well, they exhibit shorter design cycles, and they incur substantially lower costs than more tightly-coupled architectures. Furthermore, they can benefit from heterogeneity and there is potential for providing highly-available systems since component replication is not as costly as in other architectures. (ii) The success of the shared address space abstraction: Previous work [21], [1], [11] has shown that a shared address space can be provided efficiently on tightly-coupled hardware DSM systems up to the 128 processor scale. Developers have been writing new software for the shared address space abstraction and legacy applications are being ported to the same abstraction. Finally, most vendors are designing hardware cache-coherent machines, targeting both scientific as well as commercial applications.

Traditionally, shared virtual memory (SVM) clusters have been built using small-scale symmetric multiprocessors (SMPs) that follow a uniform memory access (UMA) architecture. The SAN interconnection networks used are faster than LANs and employ user-level communication that eliminates many of the overheads associated with the operating system and data copying. With current advances in technology it will soon be possible to construct commodity shared address space clusters out of larger–scale nodes connected with non-coherent networks that offer latencies and bandwidth comparable to interconnection networks used in hardware cache–coherent systems today. Nodes in these systems will be wider than today's SMPs, employing between 8 and 32 processors. At the same time, the performance characteristics of system area networks are improving. There are already SANs available that offer bandwidth comparable to that of the memory bus on today's commodity SMPs and end-to-end latencies in the order of a few microseconds [7], [14], [10]. These networks are getting closer to providing performance comparable to most tightly coupled interconnection networks that have been used in hardware cc-NUMA systems.

Although technology is changing, little is known today about how it will affect future SVM clusters or how it can be used to best improve overall system performance. In this direction, recent work [19], [5] has used system simulation to investigate future trends. These studies find that support for general purpose operations in the network interface, such as remote fetch and synchronization, can be used to reduce protocol overheads.

Our work leverages these results and provides more accurate answers to a number of questions by using detailed system emulation: (i) To what degree will future shared memory clusters benefit from using interconnection networks that offer about one order of magnitude better latencies and bandwidth than todays SANs? Are protocol costs the dominating factor, or can faster interconnection networks improve system performance? If yes, what is the achievable improvement? (ii) How does the use of wider, potentially cc-NUMA, nodes affect system performance? Can existing protocols be used in these systems, or is it necessary to adjust them to new system characteristics? How can SVM protocols best take advantage of the new architecture? (iii) What are the remaining system bottlenecks and what are the areas where further research and improvements are necessary?

To investigate these issues we build extensive emulation infrastructure. We use an existing, large–scale hardware cache–coherent system, an SGI Origin2000, to emulate future clusters. We port an existing, low–level communication layer and a shared virtual memory protocol on this system and study the behavior of a set of real applications. This communication layer, Virtual Memory-Mapped Communication (VMMC), provides near-hardware performance in exchanging messages [8]. We use VMMC to partition the system in a set of cc-NUMA nodes that communicate with message passing. On top of VMMC we provide an SVM layer that provides a shared address space abstraction across nodes. The protocol we use, *GeNIMA*, has been demonstrated to perform and scale well for a wide range of applications on a real cluster with 64 processors [4]. *GeNIMA* is already tuned for current state-of-the art SAN clusters and takes advantage of network interface support to eliminate asynchronous protocol processing and interrupts. We present results for configurations of 64-processors, which is (to our knowledge) the largest configuration used for software shared memory.

Our approach has a number of advantages. It eliminates the need for simulation and allows direct execution of the same code used on the SAN cluster on top of the emulated communication layer. Using simulation, although advantageous in some cases, tends to overlook important system limitations that may shift bottlenecks to system components that are not modeled in detail. Using emulation on top of an existing, commercial multiprocessor results in a more precise model for future clusters: It uses commercial components that will most likely be used in future nodes and a commercial, off-the-shelf operating system. In fact, using a real, commercial OS reveals a number of issues usually hidden in simulation studies and/or custom built systems. Moreover, we use an SVM protocol and communication layer that run on a Myrinet cluster and have been designed and tuned for low-latency, high-bandwidth SANs.

Our high level conclusions are: (i) System emulation is invaluable in quantifying future trends. Most importantly, however, it makes it difficult to ignore issues that may affect future systems and are inadvertently overlooked in simulation studies. (ii) Although, SVM protocols are able to take advantage of the faster interconnection network and reduce data wait time significantly, overall improvements are limited. Data wait time in many applications is reduced to less than 15% of the total execution time. With slight restructuring, the parallel speedup of FFT increases from 6 (on an existing cluster) to 23 (on the Origin2000) with 32 processors, and to 26 with 64 processors. However, barrier costs do not scale well to the 64–processor level for some applications for the problem sizes we examine. Moreover, using wide (8– and 16–processor) cc-NUMA nodes has an impact on intra-node synchronization and data placement. Most importantly, the NUMA features of the nodes result in imbalances and the increased number of processes per node results in higher page invalidation (*mprotect*) costs. Existing SVM protocols need to be adjusted to address these issues.

The rest of the paper is organized as follows: Section II presents the hardware platforms we use for this work. Section III presents our emulation infrastructure and our protocol extensions. Sections IV and V present our results. Section VI presents related work. Finally, Section VII discusses the use of emulation as a modeling method and Section VIII draws overall conclusions.

II. Hardware platform

The base system used in this study is an SGI Origin 2000 [20], containing sixty-four 300MHz R12000 processors and running Cellular IRIX 6.5.7f. The 64 processors are distributed in 32 nodes, each with 512 MBytes of main memory, for a total of 16 GBytes of system memory. The nodes are assembled in a full hypercube topology with fixed-path routing. Each processor has separate 32 KByte instruction and data caches and a 4 MByte unified 2-way set associative second-level cache. The main memory is organized into pages of 16 KBytes. The memory buses support a peak bandwidth of 780 MBytes/s for both local and remote memory accesses.

To place our results in context, we also present results from an actual, 64–processor cluster (16 Quads) that has been built [15]. A direct comparison of the two platforms is not possible due to the large number of differences between the two systems. Our intention is to use the actual cluster statistics as a reference point for what today's systems can achieve.

Each node in the cluster is an Intel PentiumPro Quad SMP, containing four 200 MHz processors, a shared snooping-coherent bus, and 512 MBytes of main memory. Each processor has separate 8 KByte instruction and data caches and a 512 KByte unified 4-way set associative second level cache. The main memory is organized into pages of 4 KBytes. The operating system is WindowsNT 4.0. Each SMP node connects to a Myrinet [7] system area network interface (NI) via a PCI bus. The 16 Myrinet NIs are connected together via a single 16-way Myrinet crossbar switch, thus minimizing contention in the interconnect. Each NI has a 33 MHz programmable processor and connects the node to the network with two unidirectional links of 160 MByte/s peak bandwidth each. Actual node-to-network bandwidth is constrained by the PCI bus (133 MBytes/s) on which the NI sits.

III. Emulation infrastructure

Supporting a programming model gives rise to a layered communication architecture that is shown in Figure 1. The lowest layer is the *communication layer,* which consists of the communication hardware and the low level communication library that provide basic messaging facilities. Next is the *protocol* layer that provides the programming model to the parallel application programmer. Finally, above the programming model or protocol layer runs the *application* itself.

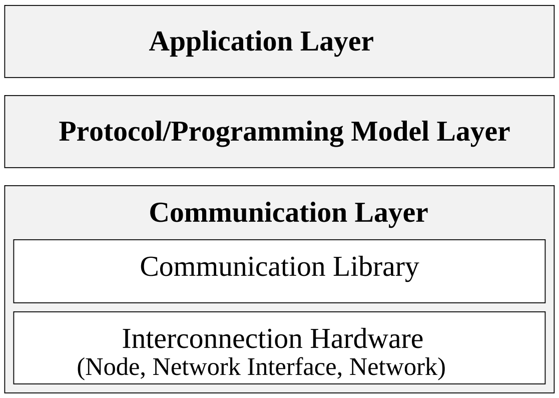


Fig. 1. The layers that effect the end application performance in software shared memory systems.

To emulate a cluster on top of an SGI Origin2000 we provide this layered architecture. We implement a user-level communication layer that has been designed for low-latency, high-bandwidth cluster interconnection networks. We then port on this communication layer an existing SVM protocol that has been optimized for clusters with low-latency interconnection networks. At the highest level we use the SPLASH-2 applications. The application versions we use include optimizations [17] for SVM systems. These optimizations are also useful for large scale hardware DSM systems.

In the next few sections we describe our experimental platform in a bottom–up fashion.

A. Communication Layer

The communication layer we use in this system is Virtual Memory Mapped Communication (VMMC) [8]. VMMC provides protected, reliable, low-latency, high-bandwidth user-level communication. The key feature of VMMC that we use is the remote deposit capability. The sender can directly deposit data into exported regions of the receiver's memory, without process (or processor) intervention on the receiving side. The communication layer has been extended to support a remote fetch operation and system-wide network locks in the network interface without involving remote processors, as described in [4]. It is important to note that these features are very general and can be used beyond SVM protocols.

We implement VMMC on the Origin2000 (VMMC-O2000) providing a message passing layer on top of the hardware cache-coherent interconnection network. VMMC-O2000 provides higher layers with the abstraction of variable-size, cache-coherent, cc-NUMA nodes connected with a non-coherent interconnect and the ability to place thread and memory resources within each node. VMMC-O2000 differs from the the original, Myrinet implementation [8] in many ways:

The original implementation of VMMC makes use of the DMA features of the Myrinet NIs: data are moved transparently between the network and local memory without the need for intervention by the local processors. The Origin2000's *Block Transfer Engine* offers similar, DMA–like services but user-level applications do not have access to this functionality. VMMC-O2000 instead uses *programmed I/O* (PIO) to transfer data between the emulated nodes. PIO operations are performed using the *bcopy(3C)* C library call. Local and remote buffers are placed on Unix shared memory regions and are visible to both the sender and the receiver.

Asynchronous send and receive operations are not implemented in VMMC-O2000. Asynchronous transmission on the cluster was managed by the dedicated processor on the Myrinet NI; the Origin offers no such dedicated unit. Although similar possibilities do exist—for instance using one of the two processors in each node as a communication processor—these are beyond the scope of this work and we do not explore them here. Asynchronous operations in the SVM protocol are replaced with synchronous ones.

VMMC-O2000 inter-node locks are implemented as ticket-based locks. Each node is the owner of a subset of the system–wide locks. At the implementation level, locks owned by one node are made available to other nodes by way of Unix shared–memory regions.

Since nodes within the Origin system are distributed cc-NUMA nodes, they do not exhibit the symmetry found in the SMP nodes that have been used so far in software shared memory systems. For this reason, we extend VMMC-O2000 to provide an interface for distributing threads and global memory within each cluster node. Compute threads are pinned to specific processors in each node. Also, global memory is placed across memory modules in the same node either in a round-robin fashion (default) or explicitly by the user. This performance difference from what today's systems can achieve, allows us to look into many aspects of future SVM clusters.

Table I shows measurements for the basic operations of both VMMC and VMMC-O2000. We see that basic data movement operations are faster by about one order of magnitude in VMMC-O2000. Notifications are asynchronous interrupts issued by VMMC to the user process on message arrival, if the user process chooses to enable them. Notification cost is about the same, but is not important in this context, as *GeNIMA* does not use interrupts [4]. The cost for remote lock operations are significantly reduced under VMMC-O2000.

VMMC Operation	SAN Cluster	Origin2000
1-word send (one-way lat)	14μs	0.11μs
1-word fetch (round-trip lat)	31μs	0.61μs
4 KByte send (one-way lat)	46μs	7μs
4 KByte fetch (round-trip lat)	105μs	8μs
Maximum ping-pong bandwidth	96 MBy/s	555 MBy/s
Maximum fetch bandwidth	95 MBy/s	578 MBy/s
Notification	42μs	47μs
Remote lock acquire	53.8μs	8μs
Local lock acquire	12.7μs	7μs
Remote lock release	7.4μs	7μs

TABLE I

Basic VMMC costs. All send and fetch operations are assumed to be synchronous, unless explicitly stated otherwise. These costs do not include contention in any part of the system.

Overall, VMMC-O2000 provides the illusion of a clustered system on top of the hardware cache-coherent Origin 2000. The system can be partitioned to any number of nodes, with each node having any number of processors. Communication within nodes is done using the hardware–coherent interconnect of the Origin without any VMMC-O2000 involvement. For instance, an 8-processor node consists of 4 Origin 2000 nodes, each with 2 processors that communicate using the hardware–coherent interconnect. Communication across nodes is performed by *GeNIMA* using explicit VMMC-O2000 operations that access remote memory. Next, we provide a brief introduction to the *GeNIMA* protocol and describe our extensions.

B. Protocol Layer

GeNIMA uses general-purpose network interface support to significantly improve protocol overheads and narrow the gap between SVM clusters and hardware DSM systems up to the 16-processor scale. The version of the protocol we use here is the one presented in [15] with certain protocol-level optimizations to enhance performance and memory scalability.

For the purpose of this work, we port *GeNIMA* on the Origin2000 (*GeNIMA*-O2000), on top of VMMC-O2000. The same protocol code runs on both WindowsNT and IRIX. This involved modifying three parts of the system: (i) thread creation, (ii) memory allocation and (iii) page fault handling. Otherwise, the core protocol code is exactly the same on both platforms. Additionally, *GeNIMA* was extended to support a 64-bit address space.

A number of architectural differences arise as a result of faster communication, the cc-NUMA nodes, and contention due to wider nodes. To address these issues we extend and optimize *GeNIMA*-O2000 as follows:

B.1 Page invalidation (*mprotect*) operations

GeNIMA uses *mprotect* calls to change the protection of pages and invalidate local copies of stale data. *mprotect* is a system call that allows user processes to change information in the process page table. Since *mprotect* is a system call, it requires entering the kernel and can be expensive. Moreover, *mprotect* calls require invalidating TLBs of processors within each virtual node, and thus the cost is affected by the system architecture. More information on the cost of *mprotect*s will be presented in Section V. *GeNIMA* tries to minimize the number of *mprotect*s by coalescing consecutive pages to a single region and changing its protection with a single call.

B.2 Data prefetching

We enhance data sharing performance by adding page prefetching to the page fault handler. When a process needs to fetch a new shared page, the subsequent N pages are also read into local memory. Empirically, we determined that N=4 offers the best prefetching performance on *GeNIMA*-O2000. Ideally, this approach can reduce both the number of `mprotect()` calls and the number of page protection faults by increasing demands on network bandwidth. The average cost of an *mprotect* call and the overhead of a page fault on the Origin2000 are relatively high: ≈150μs and ≈37μs, respectively). Thus, prefetching not only takes advantage of the extra bandwidth in the system, but alleviates these costs as well. The actual prefetch savings are dependent on the applications' particular access pattern. In order to determine when prefetching is useful, "false" prefetches are tracked in a history buffer. A prefetch is marked as "false" when the additional pages that were fetched are not used. If a page has initiated a "false" prefetch in the past, the protocol determines that the current data access pattern is sufficiently non-sequential and disables prefetching for this page the next time it is needed.

B.3 Barrier synchronization

We enhance barriers in two ways. The first change involves serializing large sections of the barrier code, while the second involves a more efficient ordering of the work to be performed.

This first change is surprisingly counter-intuitive. In the original *GeNIMA* implementation (*GeNIMA*-Myrinet) all processes within a node cooperate to perform the tasks of processing incoming diffs and updating the protection of the shared pages. One process in each node is selected as the barrier *leader*, while the others are designated as *followers*; the followers assist the leader by performing much of the barrier processing in parallel. *GeNIMA*-O2000 was changed to prohibit the followers from assisting in the processing — resulting in a complete serialization of the barrier-related work in each node. This approach eliminates concurrent *mprotect*s by processes in the same address space; this is significant for large nodes, as the cost of a single *mprotect* call roughly doubles for each additional processor issuing simultaneous requests (Figure 3). This approach is specific to *GeNIMA*-O2000, but is important for systems that will employ wide compute nodes.

The second change introduces an ordering step prior to the processing of the page invalidations. We use the *quick-sort* C library function to order the updates by page number; groups of consecutive pages are then serviced together and the entire block is updated with a single *mprotect*.

B.4 Intra-node lock synchronization

The original *GeNIMA* protocol uses the *test-and-set* algorithm to implement intra-node locks. Although this approach works well for systems with small–scale SMP nodes, it is not adequate for systems with larger–scale, cc-NUMA nodes and leads to poor caching performance and increased inter-node traffic in *GeNIMA*-O2000. We have experimented with a number of lock implementations. Overall, ticket–based locks turn out to be the most efficient. For this reason we replace these locks with a variant of the *ticket-based* locking algorithm that generates far less invalidation traffic and offers FIFO servicing of lock requests to avoid potential starvation.

C. Applications Layer

We use the SPLASH-2 [26], [16] application suite. A detailed classification and description of the application behavior for SVM systems with uniprocessor nodes is provided in [13]. 64-bit addressing and operating system limitations related to shared memory segments prevent some of the benchmarks from running at specific system configurations. We indicate these configurations in our results with 'N/A' entries. Also, we are currently trying to address some of these issues by adapting applications to 64-bit addressing and modifying *GeNIMA*-O2000 to deal with OS-imposed shared memory segment restrictions. The applications we use are:

Regular Applications: The applications in this category are FFT [2] and LU [26]. Their common characteristic is that they are optimized to be single-writer applications; a given word of data is written only by the processor to which it is assigned. Given appropriate data structures they are single-writer at page granularity as well, and pages can be allocated among nodes such that writes to shared data are almost all local. The applications have different inherent and induced communication patterns [26], [13], which affect their performance and the impact on SMP nodes.

Irregular Applications: The irregular applications in our suite are Radix [6], Volrend [22], WaterSpatial [26] and SampleSort [26]. Irregular applications exhibit more challenging data access patters for shared memory systems. Moreover, this may result in false sharing depending on the level of sharing granularity.

In this work we use both original versions of SPLASH-2 applications [26] and versions that have been restructured to improve their performance on SVM systems [16]. The same restructurings are found to be very important on large-scale hardware-coherent machines [17], so they are not specific to SVM. FFT and LU are the original SPLASH-2 applications. These versions of the applications are already optimized to use good partitioning schemes and data structures, both major and minor, for both hardware coherence and release consistent SVM. Radix, SampleSort, Volrend are the restructured applications from [16]. The version of WaterSpatial we use (WaterF) is the restructured version from [15]. The restructurings for these applications are not intrusive and try to improve data assignment, make remote accesses less scattered, or eliminate unnecessary synchronization.

Furthermore, in this work we restructure FFT (FFTst) to stagger the transpose phase among processors within each node. This allows FFT to take advantage of the additional bandwidth available in the system. Since FFTst outperforms the original version, we only present results for this optimized version.

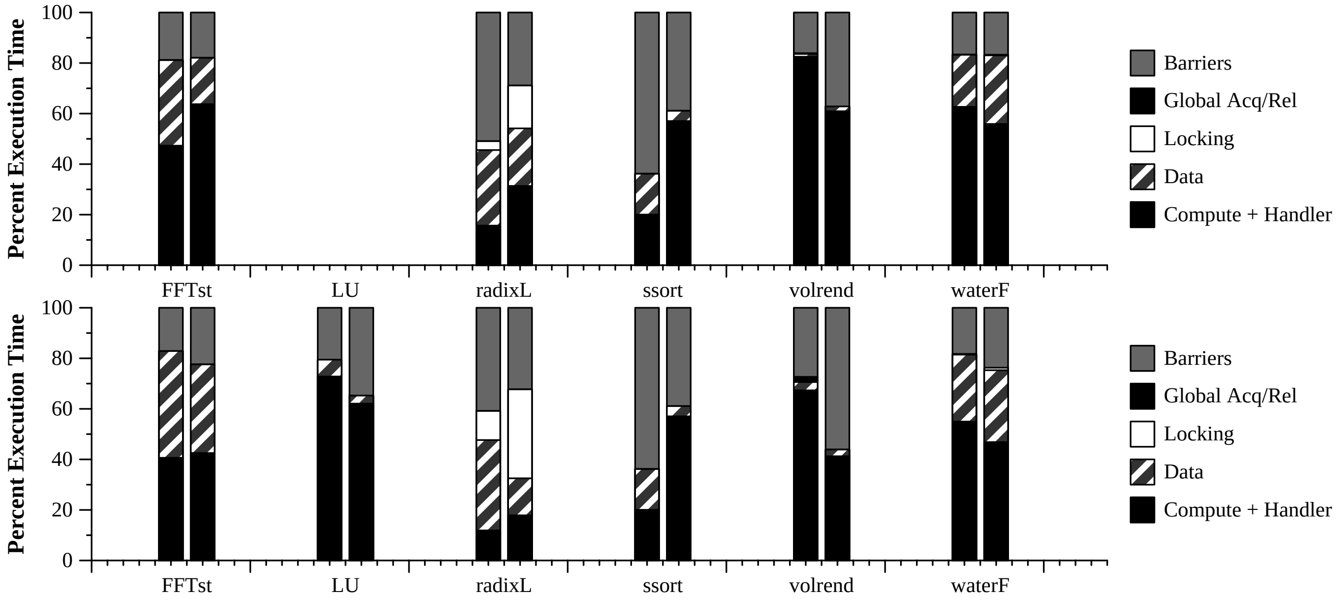


Fig. 2. Execution time breakdowns for each application. The uppermost graph provides breakdowns for a 32-processor system; the lower graph provides breakdowns for a 64-processor system. The left bar in each graph refers to *GeNIMA*-Myrinet, whereas the right bar refers to the *GeNIMA*-Origin2000. (FFT is the original version under *GeNIMA*-Myrinet and the staggered version under *GeNIMA*-O2000.)

IV. Impact of Fast Interconnection Networks

In this Section we discuss the impact of faster interconnection networks on the performance of shared virtual memory. We use system configurations with 4 processors per node for two reasons: (i) most of the work so far with SVM on clusters has used Quad SMP nodes, and (ii) we would like to place our results in context and be able to loosely relate the achievable improvements to today's systems and in particular, a previously-constructed Myrinet–based cluster [15].

In the next few paragraphs we analyze the impact of using a faster communication layer on SVM performance. Figure 2 presents execution time breakdowns for each application for *GeNIMA*-Myrinet and *GeNIMA*-O2000 for 32– and 64–processors. We divide execution time into compute, data wait, lock synchronization, and barrier synchronization time. Our goal is to investigate the impact of faster networks on future systems. Although the two platforms we use in this work cannot be compared directly due to a number of differences in the micro-processor and memory architectures they use, they offer invaluable insight into the needs of future clusters. Also, to ease the comparison and make it easier to identify the impact of the network architecture, we convert overheads and protocol costs in percentages of execution time and we use those in our investigation as opposed to absolute numbers.

A. Remote data wait

Remote data wait time is greatly improved in *GeNIMA*-O2000. This is partly due to the higher-speed communication layer and partly due to our protocol–level optimizations that take advantage of the lower latency and the higher bandwidth. More specifically:

FFTst benefits substantially from the faster network and our optimizations. By staggering the transpose phase of the original FFT we are able to take advantage of the additional bandwidth available in *GeNIMA*–O2000. The result is an impressive speedup of 23.8, compared to 6.9 for FFT with *GeNIMA*-Myrinet.

ssort produces a relatively regular pattern of data accesses and benefits most from the faster data transfer times, prefetching and the merging of *mprotect* calls: comparing its performance under *GeNIMA*-O2000 with *GeNIMA*-Myrinet, ssort generates 72% fewer page faults and the faster communication layer reduces the average service time per fault by about 65%. In absolute numbers, this represents an 80% reduction in data wait time, relative to the cluster.

radixL has a fairly high misprediction rate in prefetching, and achieves only a 30% reduction in page faults compared to the version of *GeNIMA* that does not use prefetching. Average service time per fault is increased by 30% compared to *GeNIMA*-Myrinet. This represents a 10% reduction in data wait costs, relative to the cluster. With prefetching disabled, radixL's data sharing overhead is reduced by 38% in *GeNIMA*-O2000.

LU exhibits a significant improvement in data wait time. The faster communication layer reduces the average page fault service time by 75% in *GeNIMA*-O2000 over *GeNIMA*-Myrinet. LU benefits very little from prefetching and exhibits only a 20% reduction in its total number page faults. Despite these improvements, data wait time in LU is a small component of the overall execution time, so the overall impact is negligible. Similarly, in volrend data wait time is reduced from 3.3% on *GeNIMA*-Myrinet to 2.6% on *GeNIMA*-O2000.

In waterF, data wait time increases from 24% of total execution time to 28%. This is due to the greatly reduced compute time in *GeNIMA*-O2000 as a result of the larger caches on the Origin2000. Page fetch times, however, are typically reduced by a factor of 4 or 5 over that of the cluster.

Overall, the faster communication layer and our optimizations provide significantly improved data wait performance. The direct remote read/write operations in *GeNIMA* eliminate protocol processing at the receive side (the page home) and make it easier for protocol performance to track improvements in interconnection network speed. In the majority of the applications data–wait time is reduced to at most 20% of the total execution time. Also, although a direct comparison is not possible, data wait time is substantially improved compared to the SAN cluster. Applications where prefetching is effective receive an additional benefit in reduced page-fault interrupt costs: anywhere from 30% to 80% of the total page faults are typically eliminated in prefetching, with the reduction in page faults resulting in a savings of 15% to 20% in total execution time on *GeNIMA*-O2000.

B. Barrier Synchronization

Our extensions to barrier synchronization result in moderate benefits when coupled with the faster interconnection network. Reduced latency and higher bandwidth in *GeNIMA*-O2000 result in faster synchronization at the start of barriers and reduced communication overheads. However, in some applications barrier costs do not scale well in *GeNIMA*-O2000 (as in *GeNIMA*-Myrinet) due to intra– and inter–node imbalances, as explained below. We can divide applications into three categories according to the amount of computational imbalance and their patterns of remote data access.

The barrier performance for applications that exhibit balanced computation and regular data access patters improves significantly as a percentage of the execution time compared to the cluster. FFTst demonstrates barrier costs 80% less expensive than those reported on the cluster. Similarly, ssort executes with one-third the barrier cost.

Applications that exhibit either small imbalances or irregular data access patterns incur low barrier synchronization costs (as measured in the time per barrier call). In this category fall LU, radixL, and waterF. However, barrier costs scale poorly due to the large number of *mprotect*s, the increased average cost of *mprotect*s in *GeNIMA*-O2000 and the increased wait time at larger processor counts (Figure 2). radixL best demonstrates this effect. It performs well at the 32-processor level, achieving a 50% reduction in barrier costs compared to the Myrinet cluster. With 64 processors, however, each processor is responsible for less of the total workload, and more data must be acquired from remote nodes; the resulting *mprotect* calls in the barrier releases increase the barrier cost to 1.7 times the cost observed on the cluster. Simultaneously, the average *mprotect* cost in radixL increases from 152.9μs to 325.8μs.

The final category of applications exhibit higher computational imbalances. This category includes waterF and Volrend. Although they exhibit reasonable barrier synchronization costs at lower processor counts (Figure 2), barrier costs do not scale at all at the 64-processor scale and the faster communication layer does not provide any benefit. More specifically, volrend achieves a 33% reduction in barrier costs at the 32-processor scale (Figure 2), however barrier time then increases to more than 50% of the total execution time at the 64-processor scale (Figure 2).

In summary, with the faster communication layer and our barrier–related optimizations most of the applications exhibit moderately improved barrier synchronization performance. Protocol costs are the main overhead and faster interconnects are unlikely to benefit barrier costs in future clusters, unless they are accompanied by protocol, application, and/or operating system changes to improve *mprotect* costs. The barrier cost on *GeNIMA*-O2000 for most applications is less than 30% of the total execution time. With 64 processors, the barrier cost does not scale well with the problem sizes we use here. Although increasing the problem size may alleviate these effects, it is still important to understand how future technologies may impact protocol design. Furthermore, future work should address the impact of problem size on the scalability of these overheads.

C. Lock Synchronization

Lock synchronization costs are generally affected by the cost of invalidating pages and the resulting dilation of the critical sections. For this reason, although lock acquires and releases have lower overheads in *GeNIMA*-O2000 (as shown in Table I), the higher *mprotect* costs dominate (Tables III, IV). Applications that rely heavily on lock synchronization such as radixL exhibit higher lock costs with *GeNIMA*-O2000. The locking characteristics of radixL, however, differ considerably between *GeNIMA*-Myrinet and the *GeNIMA*-O2000. Between 93% and 97% of its locks are obtained locally on *GeNIMA*-Myrinet, leading to lower overall lock times. Under *GeNIMA*-O2000, however, over 70% of its locks are remote acquires. This change is due to radixL's small critical regions (several dozen loads and stores) and the larger locking overheads (under contention) on *GeNIMA*-O2000: more time is spent acquiring and releasing the locks than is actually spent inside the critical region. The result is that the application spends a comparatively small percentage of its time inside the critical region, thus reducing the likelihood that acquires are issued while the lock is still local.

This suggests that developing techniques to limit the effect of *mprotect* cost on lock synchronization is critical for further reductions in lock synchronization overheads. Also, systems that provide more balanced *mprotect* costs would be able to take advantage of faster support for lock synchronization in the communication layer than what can currently be achieved in SANs.

	32 Proc (8x4)	**64 Proc (16x4)**
	GeNIMA-O2000	*GeNIMA*-O2000
FFTst	23.8	26.6
LU	N/A	32.9
radixL	1.9	1.0
ssort	6.7	N/A
volrend	17.8	23.1
waterF	9.2	14.5

TABLE II
Parallel speedup of benchmarks running under *GeNIMA*-O2000.

	Data Time		**Barrier Time**		**Lock Time**		*mprotect* **Time**		*diff* **Time**	
	Myrinet	O2000	Myrinet	O2000	Myrinet	O2000	Myrinet	O2000	Myrinet	O2000
FFTst	33.9%	18.3%	18.9%	17.9%	–	–	15.0%	6.1%	0.1%	0.6%
LU	N/A	N/A	N/A	N/A	–	–	N/A	N/A	N/A	N/A
radixL	29.9%	22.8%	30.9%	28.9%	3.5%	17.0%	16.7%	6.2%	18.4%	10.3%
ssort	N/A	N/A	N/A	N/A	–	–	N/A	N/A	N/A	N/A
volrend	1.1%	1.9%	16.1%	37.2%	–	–	0.5%	2.5%	0.3%	2.5%
waterF	20.6%	27.2%	16.7%	16.7%	0.1%	0.2%	14.6%	11.1%	0.2%	0.7%

TABLE III
Performance of *GeNIMA* on the SAN Cluster (Myrinet) and on the emulated system (O2000). The results are for 32-processor systems (8 nodes, 4 processors/node). Percentages indicate percent of total execution time.

D. Summary

Table II presents speedups and individual statistics for the applications run on *GeNIMA*-O2000. FFT is bandwidth–limited on the cluster; by modifying the original application to stagger the transpose phase, the speedup of FFTst is quadrupled with an impressive 23.8 on the 32-processor system. Similarly, LU performs well on both the 32– and the 64–processor configurations. volrend and waterF perform well at the 32–processor scale and, although they exhibit reasonable performance at the 64–processor scale, they do not scale as well. Finally, radixL and ssort exhibit low speedups. However, these applications do not currently scale well even on hardware cache-coherent systems [17].

Overall, the faster communication layer has a significant impact on data wait time and a smaller impact on synchronization time. Improvements in protocol data wait costs follow improvements in network speed, whereas improving synchronization requires further protocol and/or application work. Given a faster communication layer, the most important remaining protocol overhead is the cost of *mprotect* calls (Tables III, IV). The communication improvements and our protocol optimizations make communication-related costs less significant than on existing systems. In addition, it is important to note that *diff* costs are very small in *GeNIMA*-O2000, typically less than 1% and no more than 5% of the total execution time.

V. Impact of Wide, CC-NUMA Nodes

Software shared memory clusters have been traditionally built with small-scale SMP nodes that exhibit uniform memory access latencies from all processors in the node. As commodity systems of 8 to 16 processors become commercially available it becomes important to examine the impact of these wider nodes on system performance. Moreover, as processor speeds increase and transistor feature size is reduced, to cope with the increased pressure on the intra–node system interconnect, these systems may employ cc-NUMA architectures, especially at the 16–processor scale and upward. In this section we attempt to answer the question of future performance on these systems by demonstrating the performance of GeNIMA-O2000 on wide (8- and 16-processor) cc-NUMA nodes. Table V summarizes the parallel speedups for the 4-, 8- and 16-processor configurations. Table VI shows the major protocol costs as percentage of the total execution time at the 64–processor scale.

A. Local Data Placement

The NUMA effects of the nodes in *GeNIMA*-O2000 are negligible up to the 4-processor level. All processors in each node exhibit the same local memory access overheads during execution, as shown by the balanced compute time in Figure 4. However, as wider nodes are introduced, processors in each node exhibit highly imbalanced compute times.

	Data Time		**Barrier Time**		**Lock Time**		***mprotect* Time**		***diff* Time**	
	Myrinet	O2000	Myrinet	O2000	Myrinet	O2000	Myrinet	O2000	Myrinet	O2000
FFTst	42.2%	35.1%	17.1%	22.4%	–	–	6.8%	6.6%	0.4%	0.2%
LU	6.7%	3.2%	20.5%	34.8%	–	–	1.3%	1.4%	0.0%	0.3%
radixL	35.8%	14.7%	40.8%	32.3%	11.5%	35.2%	7.1%	10.2%	14.7%	4.4%
ssort	16.1%	4.1%	63.8%	38.9%	–	–	14.4%	2.1%	12.8%	0.3%
volrend	3.3%	2.6%	27.3%	56.1%	–	–	1.1%	7.0%	0.7%	0.8%
waterF	26.5%	28.4%	18.3%	23.7%	0.3%	1.0%	10.6%	11.2%	0.7%	0.8%

TABLE IV

Performance of *GeNIMA* on the SAN Cluster (Myrinet) and on the emulated system (O2000). The results are for 64-processor systems (16 nodes, 4 processors/node). Percentages indicate percent of total execution time.

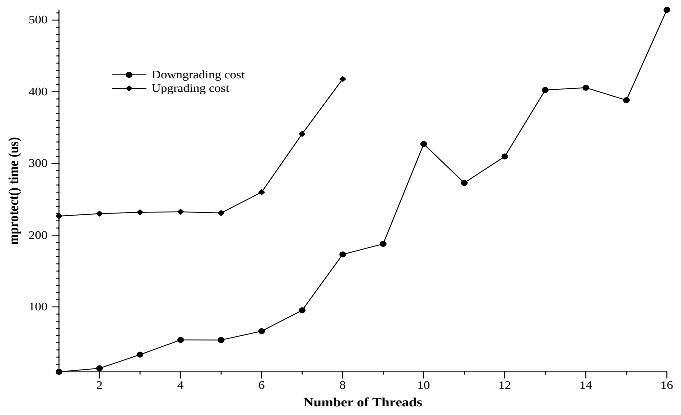


Fig. 3. Average *mprotect* cost for both upgrading and downgrading single pages. Threads in this experiment are competing with each other; although they *mprotect* different pages, they belong to the same share group. This results in TLB invalidations for all processes, as well as contention in kernel data structures.

Figure 5 shows results for FFTst on GeNIMA-O2000 with 8- and 16-processor nodes. Despite the constant problem size and workload distribution, certain processors in the wider nodes are able to complete their local computations over three times faster than others. Besides FFTst, waterF exhibits a similar behavior. This effect is not as pronounced in LU, radixL, and volrend.

To explain this behavior we need to examine what happens when pages are fetched from remote nodes. Certain processors in each node fetch more shared pages than others. Due to the programmed I/O method used to fetch pages in VMMC-Origin, new pages are placed in the cache of the processor that performs the fetch operations. Thus, due to the NUMA node architecture, processors in each node exhibit varying local memory overheads depending on the page fetch patterns. An examination of data access patterns for Figures 5 and 7 show that in FFTst processors with the lowest execution times also perform the highest number of remote data fetches. The processor that first fetches each page places data in its second–level cache, resulting in higher local memory access overheads for the rest of the processors in the same *GeNIMA*-O2000 node.

Modifying FFTst, such that processors in each node compete less for fetching pages, results in better balanced page fetches (Figure 7). Although this reduces imbalances and improves average compute time from 45% to 60% of total execution time (Figure 6), processors in each node still exhibit varying memory overheads.

Generally, applications that exhibit significant intra–node sharing of global data that are fetched from remote nodes, such as FFTst and waterF, incur highly imbalanced compute times. On the other hand, applications that either exhibit little intra–node sharing of global data, and/or have low data wait overheads overall, such as LU, volrend, and radixL, are not greatly affected. Thus, dealing with NUMA effects in wide nodes is an important problem that future SVM protocol and possibly application design has to address.

B. Remote Data Wait

In contrast to local memory access overheads, remote data wait time is reduced across applications by up to 57%. Table VI shows the percentage of execution time associated with remote data fetches, and each of the other components of the protocol overhead. The number of remote fetches are reduced on 16-processor nodes by 19% on average over 4-processor nodes. This leads to an overall reduction in remote fetch times of 15% to 20% for all benchmarks, except

	Parallel Speedup					
	32 Processors			64 Processors		
	4	8	16	4	8	16
FFTst	23.8	16.3	5.7	26.6	28.7	12.4
LU	N/A	18.5	N/A	32.9	32.5	N/A
radixL	1.9	1.8	N/A	1.0	1.2	N/A
volrend	17.8	17.7	16.1	23.1	24.8	17.6
waterF	9.2	4.9	1.3	14.5	8.5	2.7

TABLE V
Parallel speedups for 64-processor configurations with varying node widths.

	Data Time			Barrier Time			Lock Time			*mprotect* Time		
	4	8	16	4	8	16	4	8	16	4	8	16
FFTst	35.1%	21.1%	17.8%	22.4%	28.7%	19.0%	–	–	–	6.6%	9.9%	12.6%
LU	3.2%	3.2%	N/A	34.8%	35.0%	N/A	–	–	–	1.4%	0.9%	N/A
radixL	14.7%	13.5%	N/A	32.3%	31.4%	N/A	35.2%	28.2%	N/A	10.2%	4.4%	N/A
volrend	2.8%	2.6%	1.8%	56.1%	52.8%	62.1%	–	–	–	7.0%	2.3%	1.0%
waterF	28.4%	17.6%	12.2%	23.7%	31.6%	44.8%	1.0%	0.2%	0.0%	11.2%	10.7%	11.1%

TABLE VI
Protocol overhead for 64-processor configurations with varying node widths. (Percentages indicate percent of total execution time.

LU where data wait time is small and accounts only for about 3% of the total execution time.

C. Lock Synchronization

Lock synchronization also benefits from wider nodes, achieving significant reductions in overhead at the 8–processor level, and showing a moderate improvement at the 16-processor level. Local lock acquires and releases are very inexpensive in *GeNIMA* (equivalent to a few instructions). Wider nodes incur more local than remote acquires. The ratio of local to remote lock acquires changes from 0.4:1 to 1.7:1 in radixL and from 1.2:1 to 3.0:1 in waterF. All aspects of lock overhead were improved at the 8-processor level. The lock overhead in radixL was reduced from 35.2% to 28.2% of the overall execution time when moving from 4- to 8-processor nodes on GeNIMA-O2000. The average time to execute a local acquire was reduced by 32%, remote acquire time was reduced by 35% and release time was reduced by 50%. waterF exhibits similar improvements in lock overheads. However, lock time is a very small percentage of execution time, resulting in negligible impact on overall performance.

D. Barrier Synchronization

In contrast to the gains observed in remote data access and lock synchronization, barrier performance is generally unchanged at the 8-processor level. This is mainly due to slightly higher intra– and inter–node imbalances in compute times. At the 16-processor level, barrier performance is dominated by higher imbalances and high *mprotect* costs, as explained next.

E. mprotect Costs

On the cluster *mprotect* calls typically cost between 40–80μs. In contrast, the cost of downgrading a page (e.g., moving from a *read-write* to an *invalid* state in a barrier or unlock) is between 250μs and 1ms in *GeNIMA*-O2000. The cost of *mprotect* on IRIX stems from four different sources: (i) broadcasting TLB invalidations to multiple processes in the same share group; (ii) changing the protection on large portions of the address space, necessitating the modification of multiple page table entries; (iii) changing the protection of a page to a state that differs from its neighbors, resulting in the breaking of the larger protection region into three smaller ones; and, (iv) locking contention in the kernel while executing multiple, unrelated processes (in our case, in *mprotect* code). Although we ensure that (i)–(iii) are minimized in *GeNIMA*-O2000, *mprotect* cost remains high compared to the cluster. This leads us to suspect that (iv) is the reason for the additional overhead. Figure 3 shows the average cost as a function of the number of processors issuing *mprotect* calls[1] We see that the overhead for downgrading (invalidating) pages increases with the number of processors. This

[1].

	Avg. *mprotect* Cost		
	4	8	16
FFTst	127.8μs	199.5μs	622.1μs
LU	55.4μs	43.6μs	N/A
radixL	325.8μs	296.4μs	N/A
volrend	404.2μs	402.5μs	569.4μs
waterF	87.1μs	163.4μs	674.9μs

TABLE VII

Average *mprotect* cost for 64-processor configurations with varying node widths.

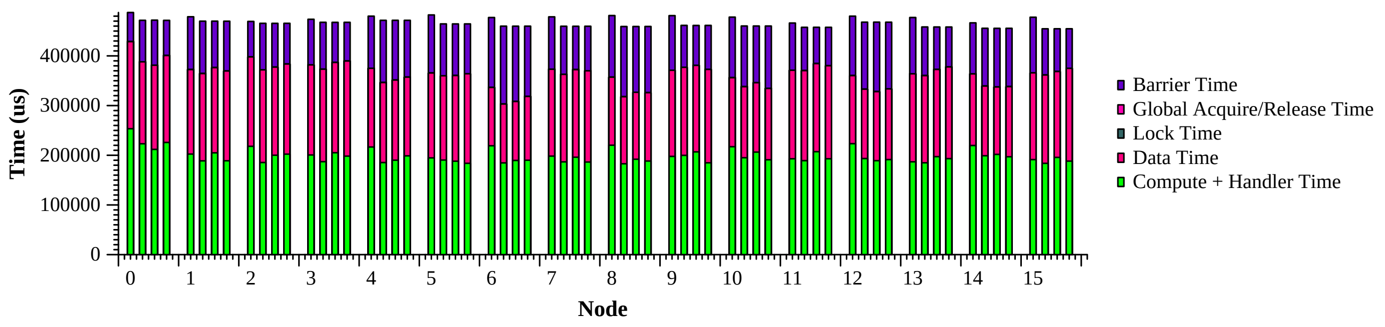


Fig. 4. Breakdown of protocol overhead in FFTst. (4 processors per node.)

aspect however of *GeNIMA*-O2000, bears further investigation.

F. Summary

Overall, although data wait time and lock synchronization overheads are reduced, compute time imbalances due to the NUMA node architecture and *mprotect* cost due to increased TLB invalidation overheads and OS contention in wider nodes, result in either small application performance improvements with 8-processor nodes, or significant performance degradation with 16-processor nodes. In building future systems with wide, cc-NUMA nodes, SVM protocols and/or applications need to address issues in intra–node data access patterns and data placement. Moreover, these effects are important even with the relatively small-scale nodes we examine here. Future clusters could support cc-NUMA nodes with even wider nodes making these effects more significant.

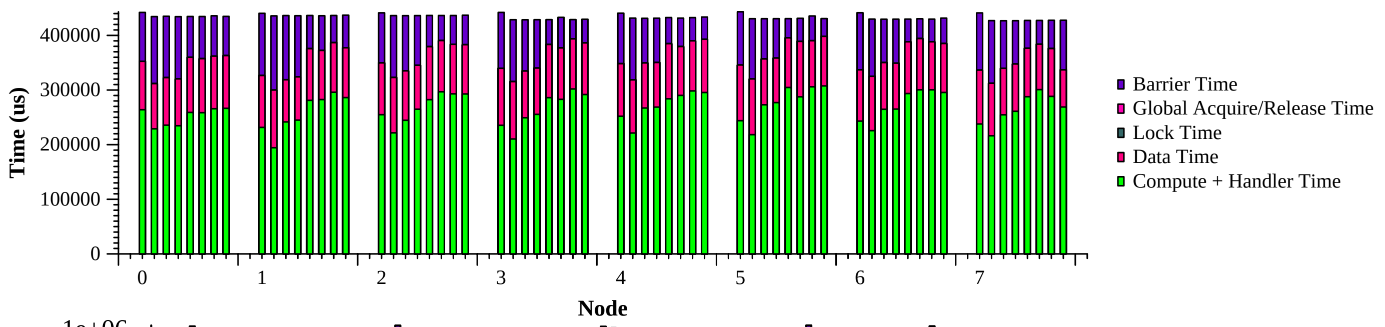
VI. Related Work

The MGS system [28] examined clustering issues for SVM systems and in particular different partitioning schemes on Alewife, a hardware cache-coherent system [1]. Unlike our work, the authors use a TreadMarks-like protocol and they find both synchronization and data movement across nodes to be a problem. The SVM protocol they use is tuned for traditional clusters with slow interconnection networks (they make use of interrupts and assume high overheads in communication initiation).

The SoftFLASH system [9] provided a sequentially consistent software shared memory layer on top of 8–processor SMP nodes. Similarly to our work, they find that the cost for page invalidations within each node is an important protocol overhead. However, they study a sequentially consistent protocol that has been used in hardware DSM systems. Our work focuses on SVM protocols that have been tuned for SAN clusters.

The authors in [19] take advantage of direct remote access capabilities of system area networks to address communication overheads in software shared memory protocols. This study is simulation-based and the performance characteristics of the interconnection network correspond to todays' state-of-the-art SANs. The focus is on protocol-level issues for extending the Cashmere protocol to clusters of SMPs and interconnection network with direct remote memory access capabilities. A subset of the features examined in the simulation studies was implemented on an actual cluster [25], which, however, corresponds more to state of the art clusters that can be built with today's technology.

Another study examined the all-software home-based HLRC and the original Treadmarks protocols on a 64-processor Intel Paragon multiprocessor [29]. This study focused on the ability of a communication coprocessor to overlap protocol processing with useful computation in the two protocols but it also compared the protocols at this large scale. However, the architectural features and performance parameters of the Paragon system and its operating system are quite different from those of today's and future clusters. Also, the study used mostly simple kernels with little computational demand,



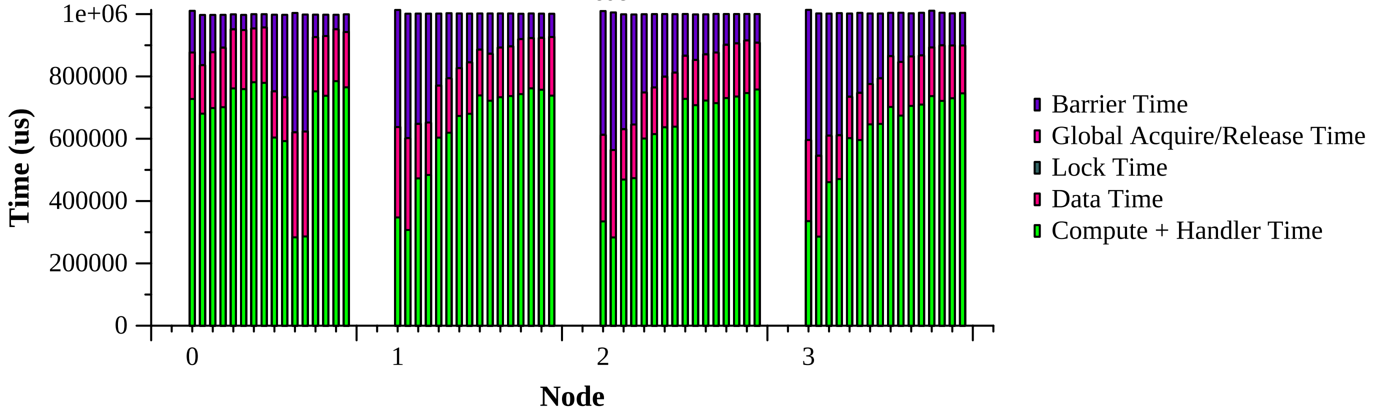


Fig. 5. Breakdown of protocol overhead in FFTst, before balancing. (8 and 16 processors per node.)

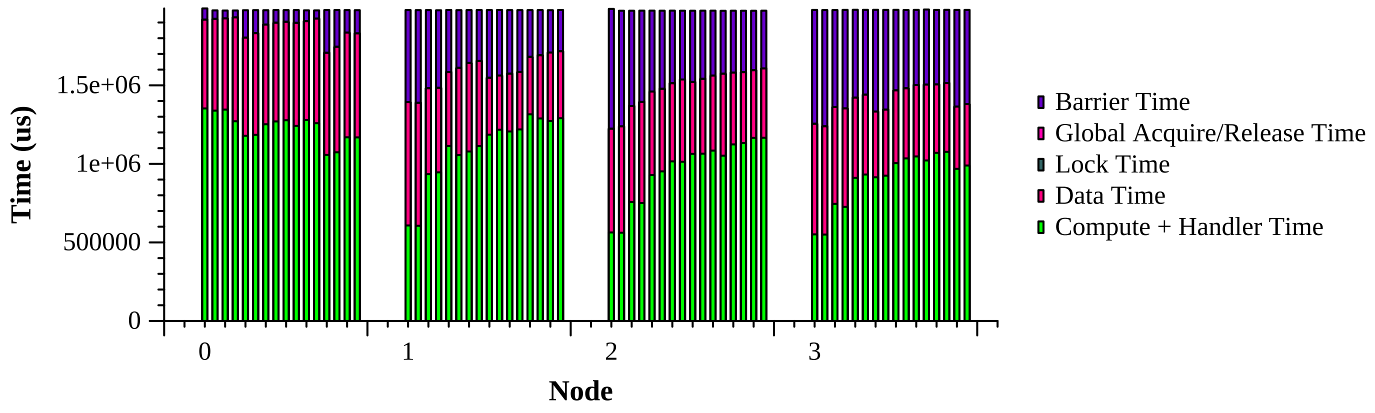


Fig. 6. Breakdown of protocol overhead in FFTst, after balancing. (16 processors per node.)

and only two real applications (WaterNsquared and Raytrace).

The authors in [5] examine the effect of communication parameters on shared memory clusters. They use architectural simulation to investigate the impact of host overhead, bandwidth, network occupancy, and interrupt cost on end-system performance. However, they only examine systems with SMP nodes. Moreover, due to the infrastructure they use, they are limited to examining relatively small problem sizes.

Overall, while a lot of progress has been made, the impact of next generation SANs and wider nodes on shared memory clusters has not been adequately addressed. This work has addressed some of the related issues in this direction.

VII. Discussion

In this paper we examine a set of questions related to building future clusters that support a shared memory abstraction. One of the most important aspects is the use of emulation as opposed to other modeling methods, mainly simulation and actual implementation, that have been used in the area. The initial motivation for using emulation was the ability to execute the same SVM protocol that we were using on the actual system and the ability to observe the effects of the operating system and the microprocessor/node architecture, since these are usually not modeled in

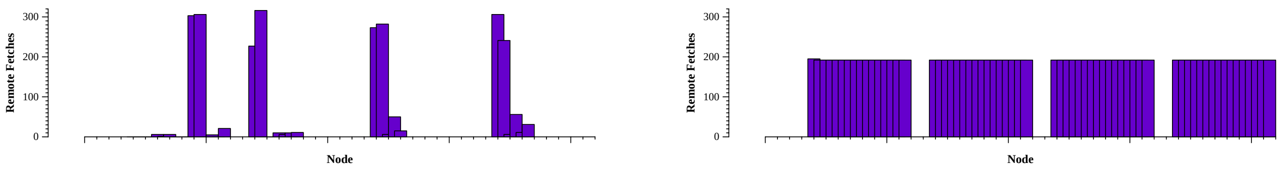


Fig. 7. Distribution of page faults, by processor, in FFTst before (left) and after (right) balancing. (4 nodes, 16 processors per node.)

simulators used in the area[2]. Of course this type of emulation, assume the existence of an expensive hardware cache–coherent system with certain required features, such as an interconnection network that offers one order of magnitude better performance than today's SANs and the ability to change the degree of clustering by changing the number of processors in each virtual cluster node. In retrospective, the use of system emulation revealed a number of issues that have been overlooked in simulation studies. Mainly, although data wait time improves significantly, other effects limit overall performance improvements and require further research in protocol design and implementation:

As commodity hardware cache–coherent systems employ more processors (8-way SMPs and even cc-NUMA), the effects of the operating system and in particular the cost of *mprotect*s is more significant than on current clusters that employ small scale SMP nodes.

Intra–node, protocol synchronization requires a lot more attention when wide nodes are used. So far, simple synchronization schemes such atomic increment operations, etc. have been adequate to implement intra–node synchronization. With increased contention, however, from multiple processors, there is a need for protocols that minimize intra–node contention on the memory bus.

Inter–node data transfer methods in SANs become important. In today's SANs, both programmed I/O [23] and DMA transfers [8] are used successfully to provide high bandwidth. However, programmed I/O may interfere with local processor caches and may result in highly imbalanced compute times among processors in the same node. Furthermore, data placement in wide system nodes, and especially in cc-NUMA nodes, results in high synchronization and compute time imbalances. Current SVM protocols do not deal with intra–node data placement and will need to be adjusted.

As node architectures in future commodity clusters become more aggressive, intra–node tradeoffs change. For instance, our results reveal a very different intra–node behavior in terms of cache misses, pressure to the memory bus, and compute times among the actual system and the emulated one. This implies that more aggressive node designs in future systems, although they will benefit overall system performance, they may require redesigning SVM protocols, e.g. to overlap protocol processing and data transfers in different ways.

All these effects result in the observation that although data wait time is reduced significantly, to at most 20% of the total execution time across all applications we examine, the overall performance improvement is less significant. Thus, although using faster networks and wider nodes offers the promise for improved performance in future clusters, today's SVM protocols do not seem to be adequate for these systems.

VIII. Conclusions

Recent and current advances in technology will make it possible in the near future to build commodity clusters out of nodes with more processors than what is available in today's SMPs. Moreover, SANs continue to approach the performance of more aggressive interconnection networks used in hardware cache-coherent systems and they will soon achieve comparable levels of performance by providing bandwidth in the order of 500 MBytes/s and μs–level latencies.

Previous simulation studies [19], [5] have shown that these advancements in technology and in particular novel architectural features, such as remote fetch operations in system area networks, can improve the performance of existing SVM clusters by supporting new protocols. In this paper we use detailed system emulation to model future clusters and study their behavior as commodity components, nodes and interconnects, improve in performance. We develop extensive emulation infrastructure using an aggressive hardware cache-coherent system and perform detailed measurements to investigate the impact of both faster interconnection networks and wider compute nodes on SVM performance. We first port a shared memory protocol that has been optimized for low-latency, high-bandwidth system area networks on a 64–processor Origin 2000. We then provide a number of optimizations that take advantage of the faster interconnection network.

Our approach eliminates the need for simulation and allows direct execution of the same code used on the SAN cluster on top of the emulated communication layer. Although system simulation is invaluable in identifying trends, we find that system emulation reveals many issues that can easily be overlooked in simulation studies. Overall, software shared memory can benefit from faster interconnection networks only if existing SVM protocols are adjusted to the architectural

[2]The large number of nodes and the existence of complex SVM protocols makes it prohibitively expensive to include detailed simulation models of system processors and operating systems.

features of future commodity components. Our work quantifies these effects and identifies the areas where more research is required for future SVM clusters.

Our results show that SVM protocols can only partly take advantage of faster communication layers. In the applications we examine, data wait time is typically reduced to less than 15% of the total execution time. Moreover, certain bandwidth intensive applications that were performing adequately or even poorly on current SAN clusters can achieve higher speedups; In particular, FFT, after restructuring that takes advantage of the additional system bandwidth, achieves an impressive 23.6 speedup on 32 processors. Barrier costs do not scale well to the 64–processor level for some applications for the problem sizes we examine. At the 32–processor scale, barrier overheads are less than 30% of the execution time, whereas at the 64–processor scale these costs increase to 40%. Finally, lock synchronization costs are not able to benefit from the faster lock acquire and release times, due to the increased cost of *mprotect* calls.

Using wide (8– and 16–processor) cc-NUMA nodes has an impact on intra-node synchronization and data placement. Most importantly, the NUMA features of the nodes result in imbalances and the increased number of processes per node results in higher *mprotect* costs. These effects combine to reduce system performance and need to be addressed in protocol design and implementation. In the absence of these effects, wider nodes offer slight improvements in remote data wait time and synchronization costs.

IX. Acknowledgments

We would like to thank Dongming Jiang for providing the modified versions of the SPLASH-2 applications and Bill Wichser for his help with using the SGI Origin 2000. We thankfully acknowledge the support of Natural Sciences and Engineering Research Council of Canada, Canada Foundation for Innovation, Ontario Innovation Trust, the Nortel Institute of Technology, and Communications and Information Technology Ontario.

References

[1] A. Agarwal, B.-H. Lim, D. Kranz, and J. Kubiatowicz. April: A processor architecture for multiprocessing. In *Proc. of the 17th International Symposium on Computer Architecture (ISCA17)*, pages 104–114, May 1990.

[2] D. H. Bailey. FFTs in External or Hierarchical Memories. *Journal of Supercomputing*, 4:23–25, 1990.

[3] R. Bianchini, L. Kontothanassis, R. Pinto, M. D. Maria, M. Abud, and C. Amorim. Hiding communication latency and coherence overhead in software dsms. In *Proc. of The 7th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS7)*, Oct. 1996.

[4] A. Bilas, C. Liao, and J. P. Singh. Using network interface support to avoid asynchronous protocol processing in shared virtual memory systems. In *Proc. of the 26th International Symposium on Computer Architecture (ISCA26)*, May 1999.

[5] A. Bilas and J. P. Singh. The effects of communication parameters on end performance of shared virtual memory clusters. In *Proc. of The 1997 Supercomputing Conference on High Performance Networking and Computing (SC96)*, Nov. 1997.

[6] G. E. Blelloch, C. E. Leiserson, B. M. Maggs, C. G. Plaxton, S. J. Smith, and M. Zagha. A comparison of sorting algorithms for the connection machine CM-2. In *Proc. of the 1st Annual ACM SIGPLAN Symposium on Parallel Algorithms and Architectures (SPAA91)*, pages 3–16, July 1991.

[7] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W. Su. Myrinet: A gigabit-per-second local area network. *IEEE Micro*, 15(1):29–36, Feb. 1995.

[8] C. Dubnicki, A. Bilas, Y. Chen, S. Damianakis, and K. Li. VMMC-2: efficient support for reliable, connection-oriented communication. In *Proc. of The 1997 IEEE Symposium on High Performance Interconnects (HOT Interconnects V)*, Aug. 1997. A short version of this appears in IEEE Micro, Jan/Feb, 1998.

[9] A. Erlichson, N. Nuckolls, G. Chesson, and J. Hennessy. SoftFLASH: analyzing the performance of clustered distributed virtual shared memory. In *Proc. of The 7th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS7)*, pages 210–220, Oct 1996.

[10] Giganet. Giganet cLAN family of products. http://www.emulex.com/products.html, 2001.

[11] R. Grindley, T. Abdelrahman, S. Brown, S. Caranci, D. Devries, B. Gamsa, A. Grbic, M. Gusat, R. Ho, O. Krieger, G. Lemieux, K. Loveless, N. Manjikian, P. McHardy, S. Srblijic, M. Stumm, Z. Vranesic, and Z. Zilac. The NUMAchine Multiprocessor. In *Proc. of the 1900 International Conference on Parallel Processing (ICPP00)*, Toronto, Canada, Aug. 2000.

[12] L. Iftode, C. Dubnicki, E. W. Felten, and K. Li. Improving release-consistent shared virtual memory using automatic update. In *Proc. of The 2nd IEEE Symposium on High-Performance Computer Architecture (HPCA2)*, Feb. 1996.

[13] L. Iftode, J. P. Singh, and K. Li. Understanding application performance on shared virtual memory. In *Proc. of the 23rd International Symposium on Computer Architecture (ISCA23)*, May 1996.

[14] InfiniBand Trade Association. Infiniband architecture specification, version 1.0. http://www.infinibandta.org, Oct. 2000.

[15] D. Jiang, B. Cokelley, X. Yu, A. Bilas, and J. P. Singh. Application scaling under shared virtual memory on a cluster of smps. In *Proc. of the 13th ACM International Conference on Supercomputing (ICS99)*, pages 165–174, June 1999.

[16] D. Jiang, H. Shan, and J. P. Singh. Application restructuring and performance portability across shared virtual memory and hardware-coherent multiprocessors. In *Proc. of The 1997 ACM Symposium on Principles and Practice of Parallel Programming (PPoPP97)*, June 1997.

[17] D. Jiang and J. P. Singh. Does application performance scale on cache-coherent multiprocessors: A snapshot. In *Proc. of the 26th International Symposium on Computer Architecture (ISCA26)*, May 1999.

[18] L. I. Kontothanassis, G. Hunt, R. Stets, N. Hardavellas, M. Cierniak, S. Parthasarathy, W. Meira, Jr., S. Dwarkadas, and M. L. Scott. VM-based shared memory on low-latency, remote-memory-access networks. In *Proc. of the 24th Annual Int'l Symp. on Computer Architecture (ISCA'97)*, pages 157–169, June 1997.

[19] L. I. Kontothanassis and M. L. Scott. Using memory-mapped network interfaces to improve t he performance of distributed shared memory. In *Proc. of The 2nd IEEE Symposium on High-Performance Computer Architecture (HPCA2)*, Feb. 1996.

[20] J. P. Laudon and D. Lenoski. The SGI Origin2000: a scalable cc-numa server. In *Proc. of the 24th International Symposium on Computer Architecture (ISCA24)*, June 1997.

[21] D. Lenoski, J. Laudon, K. Gharachorloo, A. Gupta, J. Hennessy, M. Horowitz, and M. Lam. Design of the Stanford DASH multiprocessor. Technical Report CSL-TR-89-403, Stanford University, December 1989.

[22] J. Nieh and M. Levoy. Volume rendering on scalable shared-memory MIMD architectures. In *Proc. of the Boston Workshop on Volume Visualization*, Oct. 1992.

[23] S. Pakin, M. Buchanan, M. Lauria, and A. Chien. The Fast Messages (FM) 2.0 streaming interface. Usenix'97, 1996.

[24] R. Samanta, A. Bilas, L. Iftode, and J. P. Singh. Home-based svm protocols for smp clusters: Design, simulations, implementation and performance. In *Proc. of The 4th IEEE Symposium on High-Performance Computer Architecture (HPCA4)*, Feb. 1998.

[25] R. Stets, S. Dwarkadas, N. Hardavellas, G. Hunt, L. Kontothanassis, S. Parthasarathy, and M. Scott. Cashmere-2L: Software Coherent Shared Memory on a Clustered Remote-Write Network. In *Proc. of the 16th ACM Symp. on Operating Systems Principles (SOSP-16)*, Oct. 1997.

[26] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 programs: Characterization and methodological considerations. In *Proc. of the 22nd International Symposium on Computer Architecture (ISCA22)*, pages 24–36, Santa Margherita Ligure, Italy, June 1995.

[27] D. Yeung, J. Kubiatowicz, and A. Agarwal. MGS: a multigrain shared memory system. In *Proc. of the 23rd International Symposium on Computer Architecture (ISCA23)*, May 1996.

[28] D. Yeung, J. Kubiatowicz, and A. Agarwal. Multigrain shared memory. *ACM Transactions on Computer Systems*, 18(2):154–196, May 2000.

[29] Y. Zhou, L. Iftode, and K. Li. Performance evaluation of two home-based lazy release consistency protocols for shared virtual memory systems. In *Proc. of the Second USENIX Symposium on Operating Systems Design and Implementation (OSDI96)*, Oct. 1996.