

Shared Virtual Memory Clusters with Next-generation Interconnection Networks and Wide Compute Nodes

Courtney R. Gibson and Angelos Bilas

Department of Electrical and Computer Engineering
University of Toronto
Toronto, Ontario M5S 3G4, Canada
{gibson,bilas}@eecg.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 in hardware within nodes.

In this work we investigate this approach to building future software shared memory clusters. We use an existing, large-scale hardware cache-coherent system with 64 processors to emulate a future cluster. We present results for both 32- and 64-processor system configurations. We quantify the effects of faster interconnects and wide, NUMA nodes on system design and identify the areas where more research is required for future SVM clusters. We find that current SVM protocols can only partially take advantage of faster interconnects and they need to be adjusted to the new system features. In particular, unlike in today's clusters that employ SMP nodes, improving intra-node synchronization and data placement are key issues for future clusters. Data wait time and synchronization costs are not major issues, when not affected by the cost of page invalidations.

1 Introduction

Recently, there has been a lot of work on providing a shared address space abstraction on clusters of commodity workstations interconnected with low-latency, high-bandwidth system area networks (SANs). The motivation for these efforts is two-fold. First, system area network (SAN) clusters have a number of appealing features: They follow technology curves well since they are composed of commodity components. They exhibit shorter design cycles and lower costs than tightly-coupled multiprocessors. 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. Second, the success of the shared address space abstraction: Previous work [13, 6] has shown that a shared address space can be provided efficiently on tightly-coupled hardware DSM systems up to the 128 processor scale and most vendors are designing hardware cache-coherent machines, targeting both scientific as well as commercial applications.

Traditionally, SAN clusters have been built using small-scale symmetric multiprocessors (SMPs) that follow a uniform memory access (UMA) architecture. The 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 be possible in the near future 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. With these changes in technology it is important to examine the impact that both the interconnection network performance and the node architecture have on the performance of shared memory clusters. The design space for these next generation SANs is not yet fixed; there is a need to determine the required levels of both functionality and performance. Similarly, it is not clear how using wider nodes will impact software shared memory protocol design and implementation as well as system performance.

In this paper we address the following questions: (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 today's 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 such a cluster. We port an existing, low-level communication layer and a shared virtual memory (SVM) protocol on this system and study the behavior of a set of real applications.

Our high level conclusion is that existing SVM protocols need to be adjusted to the new system features. In particular, unlike in today's clusters that employ SMP nodes, improving intra-node synchronization and data placement are key issues for future clusters. Data wait time and synchronization costs are not major issues, when not affected by the cost of page invalidations. More specifically: (i) SVM protocols are able to take advantage of the faster interconnection network and reduce data wait time significantly. Thus, interconnection network overhead is still an important factor on today's clusters. 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. (ii) 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. Overall, these effects combine to reduce system performance and need to be addressed in protocol design and implementation. (iii) The most important remaining protocol overheads are the cost of *mprotects* and intra-node imbalance generated by data placement, which affect mainly synchronization overheads.

The rest of the paper is organized as follows: Section 2 presents our emulation infrastructure and our protocol extensions. Sections 3 and 4 present our results. Section 5 presents related work. Finally, Section 6 draws overall conclusions.

2 Emulation infrastructure

To examine the behavior of next-generation clusters we use a state-of-the-art hardware cache-coherent DSM system, an SGI Origin 2000 to emulate future cluster ar-

chitectures. 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, it allows us to use an SVM protocol and communication layer that have been designed and tuned for low-latency, high-bandwidth SANs and run on a Myrinet cluster.

To emulate a cluster on top of an SGI Origin2000 we follow a 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 versions we use include optimizations [10] for SVM systems (not necessarily clustered 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.

2.1 Hardware platform

The base system used in this study is an SGI Origin 2000 [12], 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 cluster that has been built [8]. 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.

2.2 Communication Layer

The communication layer we use is Virtual Memory Mapped Communication (VMMC) [3]. 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 [1]. 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 applications with the abstraction of variable-size, cache-coherent nodes connected with a non-coherent interconnect and the ability to place thread and memory resources within each cc-NUMA node. VMMC-O2000 differs from the the original, Myrinet implementation [3] 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 Unix shared-memory regions to accomplish the sharing of memory between the sender and receiver, and *bcopy(3C)* to transfer data between the emulated nodes.

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.

Table 1 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. Notification cost is about the same, but is not important in this context, as *GeNIMA* does not use interrupts. The cost for remote lock operations are significantly reduced under VMMC-O2000.

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

Table 1. 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 explicit VMMC-O2000 operations that access remote memory.

2.3 Protocol Layer

We use *GeNIMA* [1] as our base protocol. *GeNIMA* uses general-purpose network interface support to significantly improve protocol overheads and narrow the gap between SVM clusters and hardware DSM systems at the 16-processor scale. The version of the protocol we use here is the one presented in [8] 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. Additionally, we extend *GeNIMA* to support a 64-bit address space. A number of architectural differences that arise as a result of faster communication, cc-NUMA nodes, and contention due to wider nodes. To address these issues we extend and optimize *GeNIMA*-O2000 as follows:

page invalidation (mprotect) operations: *GeNIMA* uses *mprotect* calls to change the protection of pages and invalidating local copies of stale data. *mprotect* is a system call that allows user processes to change information in the process page table 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 *mprotects* will be presented in Section 4. *GeNIMA*-O2000 tries to minimize the number of *mprotects* by coalescing consecutive pages to a single region and changing its protection with a single call.

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: $\approx 150\mu s$ and $\approx 37\mu s$, 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.

Barrier synchronization: We enhance barriers in two ways. The first change involves serializing large sections of the barrier code to reduce *mprotect* contention. The

second change introduces an ordering step prior to the processing of the page invalidations to reduce the number of *mprotect* calls. We use the *quick-sort* algorithm 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*.

Intra-node lock synchronization: The original *GeNIMA* protocol uses the *test-and-set* algorithm to implement inter-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.

2.4 Applications Layer

We use the SPLASH-2 [15, 9] application suite. A detailed classification and description of the application behavior for SVM systems with uniprocessor nodes is provided in [7]. 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. The application we use are: FFT, LU, Radix, Volrend, WaterSpatial, and SampleSort. We use both original versions of SPLASH-2 applications [7] and versions that have been restructured to improve their performance on SVM systems [9]. Finally, 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.

3 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 since most of the work so far with SVM on clusters has used Quad SMP nodes and we would like to place our results in context. This allows us to loosely compare our results with an actual cluster [8]. The two platforms cannot be compared directly due to a number of differences in the micro-processor and memory architectures they use, they offer invaluable insight in the needs of future clusters. A more detailed analysis of our results is presented in [5].

Figure 1 presents execution time breakdowns for each application for *GeNIMA-Myrinet* and *GeNIMA-O2000* for 32- and 64-processors. Table 2 presents speedups and individual statistics for the applications run on *GeNIMA-O2000*. In particular, 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.

Remote data wait: 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

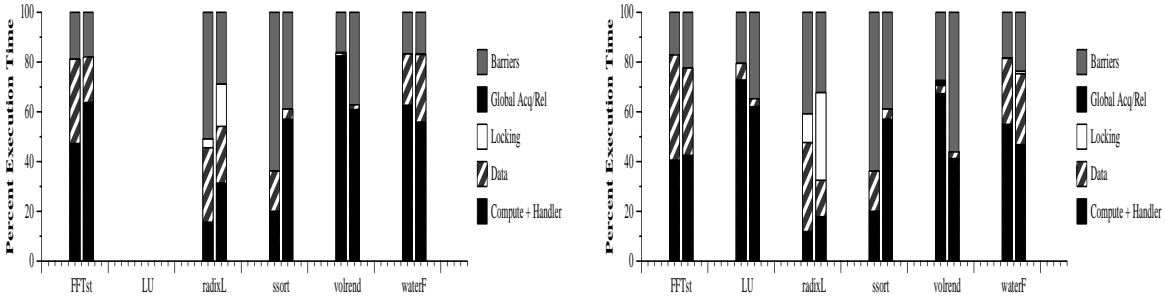


Fig. 1. Execution time breakdowns for each application. The leftmost graph provides breakdowns for a 32-processor system; the rightmost graph provides breakdowns for a 64-processor system. The left bar in each graph refers to *GeNIMA-Myrinet*, whereas the right bar refers to the *GeNIMA-Origin2000*. (FFT is the original version under *GeNIMA-Myrinet* and the staggered version under *GeNIMA-O2000*.)

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

Table 2. Parallel speedup of benchmarks running under *GeNIMA-O2000*.

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*.

Barrier Synchronization: The faster communication layer and our barrier-related optimizations result in moderately improved barrier synchronization performance for most of the applications. 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 as well 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, barrier cost does not scale well with the problem sizes we use here. Future work should address the impact of problem size on the scalability of these overheads.

Lock Synchronization: Lock synchronization costs are generally effected 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*

	Data Time		Barrier Time		Lock Time		<i>mprotect</i> Time		<i>diff</i> 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 3. Performance of *GeNIMA* on the SAN Cluster (Myrinet) and on the emulated system (O2000). The results are for for 32-processor systems (8 nodes, 4 processors/node). Percentages indicate percent of total execution time.

	Data Time		Barrier Time		Lock Time		<i>mprotect</i> Time		<i>diff</i> 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 4. 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.

(in Table 1), the higher *mprotect* costs dominate (Tables 3, 4). Applications that rely heavily on lock synchronization such as *radixL* exhibit higher lock costs with *GeNIMA*-O2000. 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 in the critical region, thus reducing the likelihood that acquires are issued while the lock is still local. Developing techniques to limit the effect of *mprotect* cost on lock synchronization is critical for further reductions in lock synchronization overheads.

4 Impact of Wide, CC-NUMA Nodes

In this section we attempt to answer the question of future performance on software shared memory clusters by demonstrating the performance of *GeNIMA*-O2000 on wide (8- and 16-processor) cc-NUMA nodes. A more detailed analysis of our results is presented in [5]. Table 5 summarizes the parallel speedups for the 4-, 8- and 16-processor configurations. Table 7 shows the major protocol costs as percentage of the total execution time at the 64-processor scale.

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

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

Table 6. Average *mprotect* cost for 64-processor configurations with varying node widths.

Local Data Placement: The NUMA effects of the nodes in *GeNIMA*-O2000 are negligible up to the 4-processor level. However, as wider nodes are introduced, processors in each node exhibit highly imbalanced compute times. 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

	Data Time			Barrier Time			Lock Time			<i>mprotect</i> 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 7. Protocol overhead for 64-processor configurations with varying node widths. (Percentages indicate percent of total execution time.

not greatly affected. 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 *bcopy* 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 in Figure 2 shows 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, reduces imbalances and improves average compute time from 45% to 60% of total execution time (Figure 2). However, processors in each node still exhibit varying memory overheads. Thus, dealing with NUMA effects in wide nodes is an important problem that future SVM protocol and possibly application design has to address.

Remote Data Wait: In contrast to local memory access overheads, remote data wait time is reduced across applications by up to 57%. Table 7 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 LU where data wait time is small and accounts only for about 3% of the total execution time.

Lock Synchronization: Lock synchronization also benefits from wider nodes, achieving significant reductions in overhead with 8-processor nodes, 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 and all aspects of lock overhead are improved with 8-processor nodes.

Barrier Synchronization: In contrast to the gains observed in remote data access and lock synchronization, barrier performance is generally unchanged with 8-processor nodes. This is mainly due slightly higher intra- and inter-node imbalances in compute times. With wide nodes, barrier performance is dominated by higher imbalances and high *mprotect* costs, as explained next.

mprotect Costs: On the cluster *mprotect* calls typically cost between 40–80 μ s. 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 250 μ s and 1ms in *GeNIMA-O2000*. The cost of *mprotect* on IRIX stems from four different sources: (i) broadcasting TLB invalidations

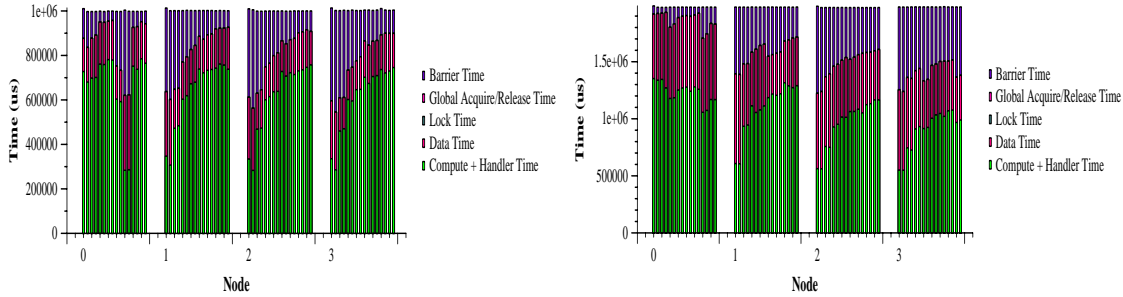


Fig. 2. Breakdown of protocol overhead in FFTst, before (left) and after (right) balancing.

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* costs remain higher the comparable costs in the cluster. This leads us to suspect that (iv) is the reason for the additional overhead¹.

5 Related Work

The MGS system [16] examined clustering issues for SVM systems and in particular different partitioning schemes on Alewife, a hardware cache-coherent system. 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 authors in [11] 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 today’s 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 [14], which, however, corresponds more to state of the art clusters that can be built with today’s technology.

The SoftFLASH system [4] provided a sequentially consistent software shared memory layer on top of 8-processor SMP nodes. They find that the cost for page invalidations within each node is very high. Another study examined the all-software home-based HLRC and the original Treadmarks protocols on a 64-processor Intel Paragon multiprocessor [17]. 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

¹ To verify this, we perform a set of experiments that are somewhat intricate so we do not describe them fully. However, an effort was made to try and capture the desired system behavior.

mostly simple kernels with little computational demand, and only two real applications (WaterNsquared and Raytrace).

The authors in [2] 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 4-way 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.

6 Conclusions

In this paper we examine the implications of building software shared memory clusters with interconnection networks that offer latency and bandwidth comparable to hardware cache-coherent systems and wide, cc-NUMA nodes. We use an aggressive hardware cache-coherent system to investigate the impact of both of these architectural features 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 results show that improvements in protocol data wait costs follow improvements in network speed, whereas improving synchronization requires further protocol and/or application work. The communication improvements and our protocol optimizations make communication-related costs less significant than on existing clusters. In addition, *diff* costs are very small in *GeNIMA-O2000*, typically less than 1% and no more than 5% of the total execution time. Barrier synchronization costs at the protocol level benefit from the faster interconnection network and the related optimizations but they do not scale well to the 64-processor level for some applications and the problems 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 to 40%. Lock synchronization costs are not able to benefit from the faster lock acquire and release times, due to the increased cost of *mprotect* calls. Finally, the most important remaining protocol overhead is the cost of *mprotect* calls.

When using wide nodes, 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 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.

Overall, software shared memory can benefit substantially from faster interconnection networks and can lead in building inexpensive shared memory systems for

large classes of applications. However, current SVM protocols can only partially take advantage of faster interconnects and do not address the issues that arise when faster networks and wider nodes are used. Our work quantifies these effects and identifies the areas where more research is required for future SVM clusters.

7 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.

References

1. A. Bilas, C. Liao, and J. P. Singh. Accelerating shared virtual memory using commodity ni support to avoid asynchronous message handling. In *The 26th International Symposium on Computer Architecture*, May 1999.
2. A. Bilas and J. P. Singh. The effects of communication parameters on end performance of shared virtual memory clusters. In *Proceedings of Supercomputing 97, San Jose, CA*, November 1997.
3. C. Dubnicki, A. Bilas, Y. Chen, S. Damianakis, and K. Li. VMMC-2: efficient support for reliable, connection-oriented communication. In *Proceedings of Hot Interconnects*, Aug. 1997.
4. A. Erlichson, N. Nuckolls, G. Chesson, and J. Hennessy. SoftFLASH: analyzing the performance of clustered distributed virtual shared memory. In *The 7th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 210–220, Oct 1996.
5. C. Gibson and A. Bilas. Shared virtual memory clusters with next-generation interconnection networks and wide compute nodes. Technical Report TR-01-01-02, Department of Electrical and Computer Engineering, University of Toronto, Toronto, Ontario M5S3G4, Canada, 2001.
6. 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 *The 2000 International Conference on Parallel Processing (ICPP2000)*, Toronto, Canada, Aug. 2000.
7. L. Iftode, J. P. Singh, and K. Li. Understanding application performance on shared virtual memory. In *Proceedings of the 23rd International Symposium on Computer Architecture (ISCA)*, May 1996.
8. D. Jiang, B. Cokelley, X. Yu, A. Bilas, and J. P. Singh. Application scaling under shared virtual memory on a cluster of smps. In *The 13th ACM International Conference on Supercomputing (ICS'99)*, June 1999.
9. D. Jiang, H. Shan, and J. P. Singh. Application restructuring and performance portability across shared virtual memory and hardware-coherent multiprocessors. In *Proceedings of the 6th ACM Symposium on Principles and Practice of Parallel Programming*, June 1997.
10. D. Jiang and J. P. Singh. Does application performance scale on cache-coherent multiprocessors: A snapshot. In *Proceedings of the 26th International Symposium on Computer Architecture (ISCA)*, May 1999.
11. L. I. Kontothanassis and M. L. Scott. Using memory-mapped network interfaces to improve the performance of distributed shared memory. In *The 2nd IEEE Symposium on High-Performance Computer Architecture*, Feb. 1996.
12. J. P. Laudon and D. Lenoski. The SGI Origin2000: a scalable cc- numa server. In *Proceedings of the 24rd Annual International Symposium on Computer Architecture*, June 1997.
13. 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.
14. 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.
15. S. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. Methodological considerations and characterization of the SPLASH-2 parallel application suite. In *Proceedings of the 23rd International Symposium on Computer Architecture (ISCA)*, May 1995.
16. D. Yeung, J. Kubiawicz, and A. Agarwal. Multigrain shared memory. *ACM Transactions on Computer Systems*, 18(2):154–196, May 2000.
17. Y. Zhou, L. Iftode, and K. Li. Performance evaluation of two home-based lazy release consistency protocols for shared virtual memory systems. In *Proceedings of the Operating Systems Design and Implementation Symposium*, Oct. 1996.