Active Memory Clusters: Efficient Multiprocessing on Commodity Clusters

Mark Heinrich, Evan Speight, and Mainak Chaudhuri

Cornell University, Computer Systems Lab, Ithaca, NY 14853, USA
{heinrich,espeight,mainak}@csl.cornell.edu
WWW home page: http://www.csl.cornell.edu/

Abstract. We show how novel active memory system research and system networking trends can be combined to realize hardware distributed shared memory on clusters of industry-standard workstations. Our active memory controller extends the cache coherence protocol to support transparent use of address re-mapping techniques that dramatically improve single-node performance, and also contains the necessary functionality for building a hardware DSM machine. Simultaneously, commodity network technology is becoming more tightly-integrated with the memory controller. We call our design of active memory commodity nodes interconnected by a next-generation network *active memory clusters*. We present a detailed design of the AMC architecture, focusing on the active memory controller and the network characteristics necessary to support AMC. We show simulation results for a range of parallel applications, showing that AMC performance is comparable to that of custom hardware DSM systems and far exceeds that of the fastest software DSM solutions.

1 Introduction

With the advent of low-cost, high-performance commodity components, networks of industry-standard workstations have captured the interest of both industry and research institutions over the past several years. Referred to as NOWs (network of workstations), COWs (collection of workstations), COPs (collection of PCs), etc., these *clusters* are typically comprised of commodity high-performance uniprocessor or small-scale symmetric multiprocessor (SMP) nodes containing two or four processors, large amounts of local memory (on the order of 1-4 GB of RAM), and a high-speed network such as Myrinet, cLAN, or ServerNet that provides zero-byte inter-node latencies less than $10\mu s$.

Clusters. As a whole, the machines comprising a cluster make up a distributed-memory parallel machine, with each node containing a complete version of the operating system, its own memory hierarchy, and I/O subsystem. This "virtual parallel machine" lends itself naturally to message passing APIs such as MPI [16]. However, many clusters are comprised of SMP nodes that already contain cache-coherent hardware, presenting the application programmer with two distinct memory interfaces: shared memory between the processors local to each machine, and distributed memory between processors that are not co-located. Because of the high overhead incurred when sending and receiving messages, there are clear performance advantages to using the native SMP load/store interface for communication between co-located processors, but accesses to remote memory then require an entirely different communication model. Utilizing two different communication models in the same program is problematic, particularly when the number of threads and processors per machine is not known at compile time.

Software DSM. In an attempt to address the programming concerns of cluster-based parallel computing, several *software DSM* systems have been built that provide shared-memory parallel programming on the cluster as a whole without requiring specialized hardware (e.g., Ivy [14], TreadMarks [8], Munin [2], Brazos [22], and Shasta [20]). These systems typically use page-level granularity to enforce coherence, manipulating the virtual memory page protections to trap protection violations and executing the coherence protocol in software on the main microprocessor. Utilizing page-level granularity also allows the high cost of communication to be amortized over a large coherence unit (4-16 KB for a typical page). Although multiple-writer protocols [2] and relaxed consistency models [2, 8] significantly reduce the amount of communication necessary to maintain coherence between nodes, high communication costs often result in poor performance.

Other software DSM systems instrument application code to check for coherence actions that need to be performed before each access to shared memory (e.g., the Shasta system [20]). These systems can then implement coherence at any granularity desired, but the high handler overhead and the fact that the network is typically integrated on the I/O bus rather than the memory bus still results in the choice of pages for data transfer. Subsequently, such systems also incur large software overhead compared to hardware DSM systems. Despite this overhead, and other factors such as high synchronization rates, frequent sharing, or large amounts of false sharing that can severely hinder the performance of software DSM systems, the cost advantages of software DSM clusters make them a viable alternative in many situations.

Hardware DSM. Alternatively, some systems provide direct hardware support for scalable shared memory, adding substantial cost to each individual node in exchange for higher performance on parallel applications. Such *hardware DSM* machines include DASH [13], the SGI Origin 2000 [12], and the Sun S3.mp [17]. Most high-performance hardware DSM machines have tightly-integrated node or memory controllers that connect the microprocessor both to the memory system and to a proprietary high-speed switching network. The scalable coherence protocols used in such machines are implemented either in hardware finite-state machines or in software running on an embedded programmable device in the controller. Despite the resulting high performance of these systems, and efforts to show that the necessary additional hardware to support hardware DSM in commodity workstations and servers is small [13], high-end PC servers and engineering workstations have yet to integrate the additional functionality needed to build seamless hardware DSM from COTS (commodity off-the-shelf) components. As discussed in Section 2, the primary reason for this is that this additional hardware does nothing to improve single-node performance.

In this paper, we show that our research in active memory systems and our subsequent active memory controller, combined with emerging network technology trends, can create the realization of hardware DSM performance on commodity clusters at the cost of previous software DSM efforts, with a concomitant improvement in single-node performance. We call the result of this convergence *active memory clusters* (AMC).

In Section 2 we explain the key differences between hardware and software DSM systems in more detail, and how our research in active memory systems narrows the gap between the two. Section 3 presents the design of active memory clusters and discusses the architectural, network, and operating systems issues in implementing AMC. Section 4 discusses the parallel performance of AMC relative to that of hardware DSM solutions, and presents simulated AMC speedup results for several parallel applications. In addition, we explore the performance impact of varying the latency and bandwidth of the commodity network interconnect. Section 5 concludes the paper.

2 Differences Between HW DSM and SW DSM

To the naïve eye, a physical comparison of a hardware DSM machine with a modern software DSM system based on clusters reveals few differences. Both machines are constructed out of individual commercial boxes connected together with proprietary high-speed networks. Closer examination reveals three main differences between the two systems:

– hardware DSM networks are faster and more tightly-integrated
– nodes in software DSM systems run separate versions of the operating system
– hardware DSM requires a specialized node controller

The first difference is the speed and integration level of the network. Typically the communication latency in software DSM networks is an order of magnitude more than in hardware DSM networks ($\sim 10\mu s$ versus under $1\mu s$). In addition, commodity motherboards integrate the network on the I/O bus versus the tighter integration on the node or memory controller in hardware DSMs. These are real differences, but they are rapidly disappearing. The computing industry's new InfiniBand network (discussed further in Section 3.2) has latencies on the order of $1\mu s$ (similar to hardware DSM latencies) and will be connected directly to the memory controller on future commodity motherboards [7].

Unlike hardware DSM systems in which every node is under the control of a single operating system, software DSM systems run in an environment where each node executes its own version of the host operating system. The critical aspect of this distinction with respect to hardware DSM is the lack of a central page table accessible by all nodes in the system. Instead, each operating system maintains its own set of virtual-to-physical mappings. As discussed in Section 3.3, we can remove this distinction by making a "distributed page table" that acts like the centralized page table in a hardware DSM-capable operating system. Alternately, a NUMA-aware OS providing a single-system image across the entire cluster can be used in the context of an AMC system.

The only remaining architectural difference is the last one listed above: the specialized node controller. In hardware DSM machines, this node controller implements the directory-based coherence protocol at a fine granularity and offloads the overhead from the main microprocessor. If these functions were integrated into the memory controller of a commodity server, there would be essentially no remaining difference between the two architectures.

While the integration of the specialized hardware DSM functions into commodity servers and workstations is possible, the economic arguments have not been compelling enough to include this functionality for two main reasons. Most importantly, this additional functionality does not improve single-node performance, meaning it is unlikely it would ever be included in commodity servers. Second, the size of the high-performance hardware DSM market has never been large enough to warrant true commodity nodes with hardware DSM support.

The debate about whether the necessary controller functionality will ever become commodity would be left at that, except for two factors. First, there is a trend toward placing more CPUs per machine in today's SMP boxes (e.g., the 32-processor SMP solution currently offered by Unisys) that is naturally accompanied by a higher cost per box. The desire to keep the cost of individual boxes low, while retaining the ability to program an entire cluster as if it were a single SMP, will be a powerful economic argument for including support for hardware shared-memory in commodity cluster components. Second, we will show how recent research into the single-node performance benefits of *active memory systems* argues for their inclusion in high-end servers, providing hardware DSM support "for free" if the active memory support is implemented in a flexible manner.

2.1 Active Memory Systems

One of the biggest challenges facing modern computer architects is overcoming the *memory wall* [19]. Technology trends dictate that the gap between processor and memory performance is widening. Though good cache behavior mitigates this problem to some extent, memory latency remains a critical performance bottleneck in modern high-performance processors. Heavily-pipelined clocked memory systems have improved memory bandwidth, but do nothing to address memory latency or reduce the number of cache misses incurred by the processor.

One approach to reducing the gap between processor and memory performance is to move processing into the memory system by using active memories [4, 5, 9, 15, 18, 23] or active memory controllers [3, 10]. In many such systems, parts of a program that have poor cache behavior are executed in the memory system, thereby reducing cache misses and memory bandwidth requirements. Other work employs address re-mapping techniques to re-structure data (like linked lists or non-unit-stride accesses) so that the processor can access them in a more cache-efficient manner. Recently, we made the observation that active memory techniques can be treated as an extension of the cache coherence protocol [10] because of the coherence problem created by address re-mapping techniques. Our active memory controller implements the coherence protocol and the extensions necessary to support active memory operations on data in the memory system, allowing the memory system to transparently and coherently access any data in the system. In this paper we detail our active memory controller architecture, its integration with emerging commodity network technology, and the resulting high-performance, low-cost, coherent DSM architecture of AMC. Our particular active memory controller approach is described in more detail in Section 3.1.

Convergence. We return now to our discussion of hardware DSM versus software DSM systems. Our active memory controller that is designed to improve uniprocessor and single-node performance contains the same functionality needed to enable cluster-based hardware DSM systems. This fact—that active memory controllers and hardware DSM controllers share much of the same functionality—lies at the heart of the AMC design. It strengthens the cases of both the active memory and hardware DSM advocates for inclusion in commodity servers. Even if the individual arguments for including specialized controller functionality fall short, their combined benefits may be enough to finally produce commodity nodes with active memory controllers.

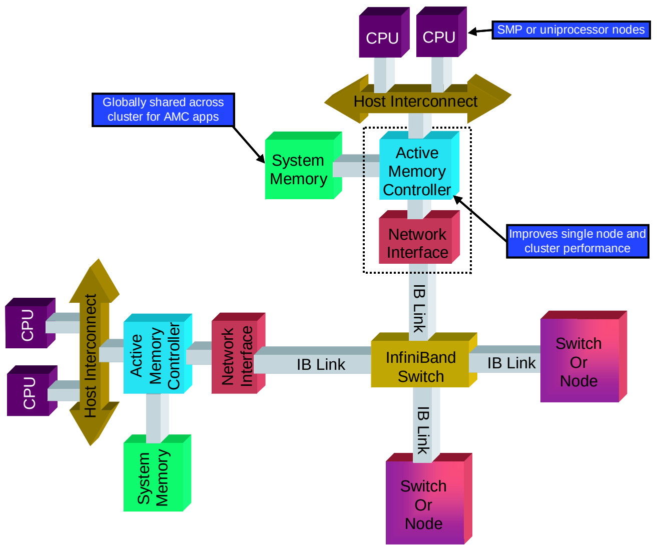

Fig. 1. Active Memory Cluster Configuration.

3 Active Memory Clusters Implementation

This section discusses the implementation details of our active memory cluster system (see Figure 1), focusing on the three architectural differences between current hardware and software DSM machines that we presented in Section 2. We first describe our active memory controller functionality, present our active memory controller design, and show how it can dramatically improve uniprocessor and single-node performance. We further explain how this functionality enables a hardware DSM machine to be constructed from cluster components. We then discuss the ramifications of upcoming tightly-integrated commodity networks and the details relevant to AMC. Finally, we end with a discussion of the system software issues related to our active memory clusters implementation.

3.1 Active Memory Controller Functionality

Recent active memory proposals have advocated the technique of re-mapping the address space of a process in an application-specific manner. Accesses to this space are then used as a signal to the memory controller to perform "active" operations rather than satisfying these accesses from physical memory [3, 10, 15]. For example, when performing matrix operations that require row and column traversals, one traversal uses the cache effectively whereas the other does not. By providing multiple memory viewpoints of the same matrix using shadow address spaces, row traversals are unchanged, whereas column traversals are treated as row traversals of a matrix at a different (shadow) address. The active memory controller fetches individual double-words from a column and returns them in a single cache line. Data is therefore provided in blocks that can be cached efficiently by the main processor. The result is good cache behavior for both row and column traversals of the matrix. Such an approach speeds up many scientific applications by using the processing capability in the memory system. Similar re-mapping techniques are used to speedup sparse matrix codes, scatter/gather operations, reductions, and linked-list-intensive programs. These, and other active memory operations are described in more detail in [10].

The main challenge with this active memory approach is solving the cache coherence problem it creates. For example, if columns of a matrix are being written via a different address space during column traversals, the next row traversal via the normal address space will return incorrect or stale data unless care is taken or costly cache flushes are performed. The key insight into solving the coherence problem in active memory systems is that the active memory controller controls both the coherence protocol *and* the fetching of the data requested by the processor. In architectures like the FLASH multiprocessor [11] and the S3.mp [17], the coherence protocol itself is programmable or extensible. Thus, active memory support is, in effect, an extension of the cache coherence protocol. In this case, the active memory controller enforces coherence between the original and shadow address spaces.

Our work with active memory controllers indicates that the occupancy of an active memory controller is significantly reduced by the introduction of a hardware unit to support the fast assembly of cache lines from individual double words (and their disassembly as well). Adding this special data unit to our flexible coherence engine reduces the occupancy of active memory operations by up to an order of magnitude, thereby improving overall system performance. At the same time, a flexible engine need not slow down non-active requests, as the processing on the AMP in Figure 2 occurs in parallel with the memory lookup needed to satisfy the cache miss [10].

Our active memory controller microarchitecture (see Figure 2) combines a flexible coherence engine with the aforementioned special data unit to support

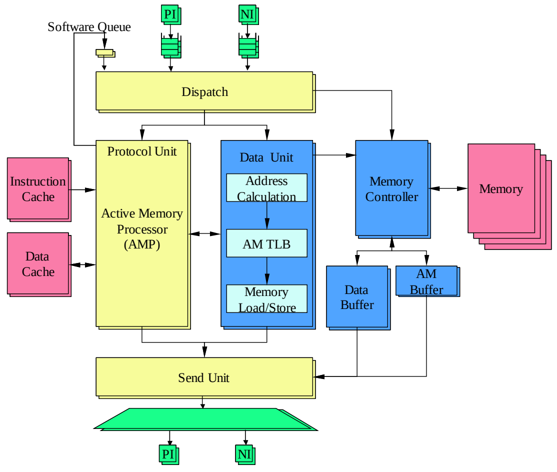

Fig. 2. Active Memory Controller Microarchitecture.

fast address re-mapping and dynamic cache line assembly. Detailed simulations of this microarchitecture can be found in [10], but we repeat representative results here. Figure 3 shows uniprocessor speedup between 1.75 and 2.8 when using our active memory controller versus systems with normal memory controllers. In these applications the number of application cache misses is reduced by over a factor of two, and prefetching the transformed shadow address space offers room for additional speedup not available in the normal application. The source code of these applications was changed minimally (on the order of 10 lines of code) to achieve these results. As shown in [10], the same architecture can also improve the performance of single-node multiprocessors via the same methods. In this paper we will further show how to extend this architecture to create multi-node DSM systems.

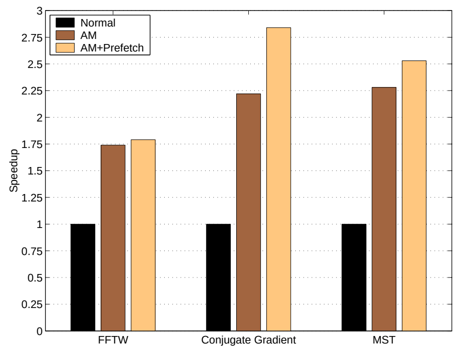

Fig. 3. Uniprocessor Active Memory Speedup.

Because we leverage cache coherence, our active memory controller design achieves this performance without requiring cache flushes on the processor. Similarly, our design is also compatible with the SMP nodes commonly used in clusters (where, because of process migration, architectures that rely on cache flushes must flush *all* the caches on the node, even for uniprocessor applications). In the remainder of this section we describe how to use this same architecture to build active memory clusters and highlight the relevant features of our microarchitecture to AMC.

Active Memory Controller Requirements. Our active memory controller manages the cache coherence protocol. Consequently, our flexible active memory controller already tracks sharing information in the system and invalidates and retrieves cache lines from the processor caches. For active memory support the processor can communicate with the memory controller through uncached writes to a portion of the address space where these writes are interpreted as commands by the memory controller (e.g. notifying the controller about shadow address spaces during initialization). An active memory controller must also have the ability to dispatch both normal and active coherence handlers by looking at certain bits in the request and address on the processor bus.

Additional support required for AMC. To support AMC, the memory controller must have a network interface as shown in Figure 1 in addition to the processor interface. In forthcoming network architectures such as InfiniBand, the network interface will be moved closer to the main CPU. We discuss the effect of the memory controller being directly connected to the network interface in more detail in Section 3.2. The additional features needed for AMC beyond those already present in the active memory controller are the ability to dispatch handlers from the network interface (including deadlock avoidance in the scheduling mechanism) and the existence of the corresponding coherence handlers for network messages. In flexible active memory controllers with a direct interface to the network, the additional coherence handlers that are needed consist of protocol code running on the memory controller (AMP in Figure 2) and do not necessitate architectural changes to the active memory controller.

3.2 Network Integration

Commodity architectures are witnessing evolutionary changes in network integration as the network connection moves from a plug-in card on the distant I/O bus to a routing chip directly connected to the memory controller (see Figure 1). Although these networks are designed primarily for use in storage networks and support user-level heavyweight protocols, our active memory controller only uses the physical routing capabilities of these directly-connected network switches. In addition, coherence messages travel between memory controllers only and are not forwarded up to the processor for handling via interrupts. Instead, an active memory cluster handles the messages entirely in the memory controller, similar to a hardware DSM machine.

Our AMC controller design uses a system area network (SAN), such as InfiniBand, as its network interface. However, the active memory controller does not use the high-level (and higher overhead) user-level protocols that run over the SAN network link when sending and receiving the messages that comprise the coherence protocol. Instead, our controller uses the fast underlying link-level performance to synthesize and handle messages to support DSM. InfiniBand supports these low-level messages via its *raw packet format*, which has an overhead of 14 bytes on top of the data payload of the packet [7]. Although this message overhead is more than that present in many hardware DSM machines, it is still small compared to the data payload (typically a cache line, or 128 bytes in our system).

The latency and bandwidth characteristics of system area networks such as InfiniBand, as well as the physical routing capabilities, are comparable to networks used in today's hardware DSM machines. InfiniBand supports virtual lanes, a critical feature for supporting deadlock-free request/reply networks in DSM machines. The "hop time" through an InfiniBand switch is currently on the order of 150 ns, and will likely drop in the near future with later generations of the hardware. The bandwidth of a single link in the InfiniBand network is 250 MB/s (2.0 Gb/s). Although this is less bandwidth than the network in an Origin 2000 (800 MB/s), it is still 2.5 times higher than that used in the most advanced current software DSM machines. Coupled with the its low latency, this network will provide an excellent base on which to build a high-performance DSM machine. We show simulation results of a prototype active memory cluster in Section 4 with varying network latencies and bandwidth parameters.

3.3 Operating System Issues

As mentioned in Section 2, software DSM runtime layers are typically used on clusters of workstations, each running their own version of the host operating system. The majority of these clusters consist of the same types of machines running the same version of the same operating system. The reasons for this include the cost advantage of buying in bulk, ease of cluster-wide system management, and the avoidance of the overhead associated with translating between different data representations. Clusters providing high-performance parallel programming environments are not groups of machines sitting on desks in an office environment, but rather sets of rack-mounted machines sequestered in machine rooms. Thus, while companies such as *Entropia* seek to use heterogenous, widely-spread machines for large-scale independent computations, realistically we do not expect high-performance cluster machines to be constructed of different components.

In AMC, as in other hardware and software DSM implementations, the virtual address of the shared global memory region must be the same on all nodes in the cluster. Furthermore, unlike software DSM, the virtual-to-physical mapping must be the same across the cluster because the AMC memory controller maintains coherence based on the physical addresses presented. With a single system image, the OS running on the cluster provides this shared page table directly for AMC. With separate operating systems, a small AMC runtime library is used to ensure the VA-to-PA mappings are consistent across the cluster during program initialization. Thus, AMC requires that the host OS support the ability to request that a specific virtual address map to a physical address within a certain range of the physical memory address space, a feature commonly used to map I/O devices into a process' virtual space. With this ability, we can create a "distributed page table" in which the shared region of memory resides at the same virtual and physical location across all OS instances. This procedure, as well as issues regarding pinning of shared pages and shared pointer distribution, are detailed in [6]. For operating systems with readily-available source (i.e., Linux or FreeBSD), we can modify the virtual memory manager to provide the necessary functionality required by AMC. Operating systems such as Windows 2000 can also provide this functionality through the use of device drivers.

Several operating systems are currently available that provide NUMA support, including Irix from SGI, Windows XP from Microsoft, and a NUMA-aware version of Linux. Any of these operating systems will run on active memory clusters with little or no modification.

4 AMC Performance Results

We simulated several parallel applications from the SPLASH-2 application suite [24] running on an AMC system connected via a SAN with InfiniBand-like characteristics: FFT with 1M points, LU decomposition with a 512x512 matrix, Ocean with a 514x514 ocean, Radix-Sort with 2M keys and a radix of 32, and Water with 1024 molecules.

Table 1. AMC hardware configuration.

Parameter	Value
Number of Nodes	1-32
Processors per Node	1
Processor Clock Speed	2.0 GHz
System Clock Speed	400 MHz (2.5 ns)
Primary Data Cache	32 KB, 32 B line size
Primary Instruction Cache	32 KB, 64 B line size
Secondary Cache	1 MB, 128 B line size
Max. Outstanding Misses	4
Network Interface Delay	Inbound: 40 ns; Outbound: 20 ns
Memory Latency from AMC Controller	125 ns to first double-word

These execution-driven simulations use the parameters described in Table 1. Our AMC simulator models our flexible active memory controller from Figure 2 connected to a network with the latency, bandwidth, and overhead characteristics of a system area network such as InfiniBand. The AMC results assume a network comprised of 16-port switches connected in a fat-tree configuration for varying network latencies. We present results for 3 different network "hop times" corresponding to current and future SAN architecture generations. Results for 450 ns show the performance with first-generation SAN parameters, which have been available for some time. The 150 ns performance numbers reflect the current state-of-the-art in system area networks, and the 50 ns results show the performance of AMC as system area networks mature. For comparison, results for a custom hardware DSM system are shown as well. The HWDSM system modeled in detail here has 12.5 ns hop times and is connected in a mesh configuration. Note that uniprocessor execution times for the HWDSM and AMC simulations are identical, indicating that speedup comparisons between the architectures is a valid metric for performance. The coherence protocol running on

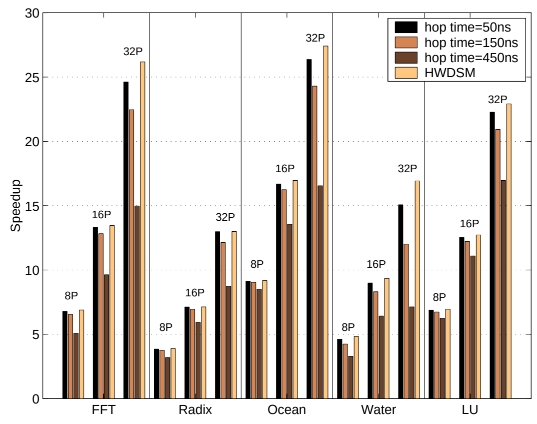

Fig. 4. AMC SPLASH-2 speedup.

the AMC controller is dynamic pointer allocation [21], a linked-list protocol that scales well to the processor counts used here. Figure 4 reports speedup on 8, 16, and 32 processor systems for the parallel execution times of the applications.

AMC performance. As shown in Figure 4, the performance of the SPLASH-2 benchmarks on AMC is remarkably good. When the interconnection network has a 50 ns hop time, all applications perform nearly as well as the custom hardware DSM implementation, despite the fact that the HWDSM architecture has a hop-time over four times faster than this AMC configuration. At the other end of the spectrum, when the network latency is 450 ns we find that AMC performs significantly worse than HWDSM. For 32-processors, HWDSM performs an average of 72% better than AMC across all 5 applications.

The bars representing 150 ns hop times most closely represent the current InfiniBand architecture, which we are using as our representative SAN. At 150 ns, HWDSM outperforms AMC by 16% for FFT, 7% for Radix-Sort, 12% for Ocean, 40% for Water, and 9.4% for LU. Even if system area networks do not ultimately achieve the 50 ns latency represented in our experiments, the bandwidth of SANs is sure to increase in subsequent generations due to the importance of I/O. For example, in the InfiniBand architecture it is possible to add additional links to increase bandwidth substantially over the 250 MB/s shown here.

To evaluate the impact of increasing the bandwidth of AMC's network, we also simulated our five applications with a network bandwidth of 1 GB/s. The results for 32 processors are shown in Figure 5. This quadrupling of the network bandwidth improved the performance of all 5 applications, most notably that of Radix, FFT, and Ocean, all of which have much higher network requirements than Water or LU. Radix shows a 46% improvement in speedup with the additional bandwidth, Ocean improves by 18%, and FFT is 14% faster.

We also expect the performance of a traditional software DSM system to improve when using future system area network architectures due to three features of emerging network architectures: a significant reduction in memory latency, the capability of remote DMA operations, and reliable communication mechanisms. Nevertheless, we still expect AMC to outperform software DSM due to software DSM's remaining high kernel overhead involved with page exception handling, the large granularity of sharing required, and the processing overhead for runtime structures such as `diffs` and `twins`. By way of comparison, the best reported software DSM speedup on 32 processors for these applications was achieved by running a software DSM system on the Origin 2000, resulting in speedups of 5.5 for FFT, 16.5 for LU, 5.9 for Ocean, and 14.1 for Water [1]. AMC achieves substantially better speedup than even this optimistic software DSM scenario.

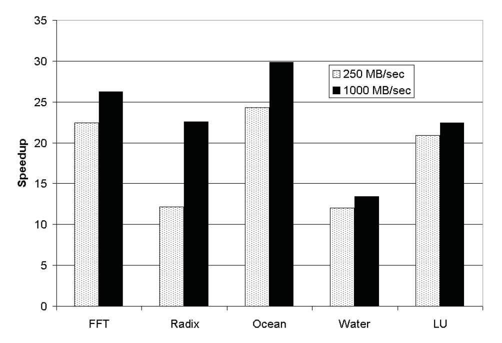

Fig. 5. AMC SPLASH-2 speedup at different network bandwidth (32 P, 150 ns hops).

Finally, we emphasize that despite the impressive results, the parallel performance of AMC does little toward strengthening the argument for putting the required AMC functionality in commodity servers. The only way that active memory controllers will be included in commodity servers is if the industry embraces the research results showing that active memory techniques like those discussed in Section 3.1 will significantly improve the performance of a standalone machine. When this happens, the insight of this paper is that designing a *flexible* active memory controller allows the realization of hardware DSM performance with commodity components using active memory clusters.

5 Conclusions

In this paper, we have shown that building clustered compute farms that deliver hardware DSM parallel performance at software DSM cost is becoming a real possibility. The only necessary changes are utilizing a more tightly-integrated network technology, adding new functionality in the memory controller, and perhaps adding a small amount of operating system support. Detailed simulations show the resulting active memory clusters architecture achieves excellent parallel performance.

Of these three enabling mechanisms, the most problematic is the additional memory controller functionality. Historically, the additional cost of changing the memory controller to support hardware DSM was not justified by the expected performance gain for two main reasons. First, the required support did nothing to help single-node performance, which is the metric by which the computer industry measures any technology for inclusion in "commodity" boxes. Second, the number of users that would benefit from such a costly change to the memory controller architecture was relatively small. However, research in active memory systems has argued for enhanced memory controller functionality to improve *single-node* performance. By treating active memory support as an extension of the cache coherence protocol, we showed that the controller support needed for active memory and hardware DSM is almost identical, provided the active memory mechanisms are implemented in a flexible manner.

Industry has already begun addressing the issue of network/system integration to support the low communication latency required by the cluster-based systems in use today. We used InfiniBand as an example of such a network architecture, but others exist that would work equally well with active memory clusters. The only remaining issue is one of system software. Software DSM machines have had a distinct advantage in this area, because the use of commodity operating systems is an important factor in keeping both initial system cost and subsequent upgrade costs low. The active memory cluster architecture can certainly be used with an operating system that natively supports hardware DSM. Alternately, simple functionality provided by device drivers can be used with commodity operating systems to achieve hardware DSM performance, even in the absence of a specialized DSM operating system.

In summary, our research in the area of active memory systems shows that active memory techniques substantially improve single-node performance in cases where caching behavior is poor. The results presented in this paper also show that this additional memory controller functionality, if implemented in a flexible manner, results in clusters of servers with the necessary components to achieve the parallel performance of a hardware distributed shared memory system. These two performance benefits are a strong argument for the inclusion of active memory systems in commodity nodes, resulting in a parallel computing platform comprised of industry-standard servers that exhibit parallel performance far surpassing that of traditional cluster and software DSM efforts.

Acknowledgments

This research was supported in part by Cornell's Intelligent Information Systems Institute, Microsoft Corporation, and NSF CAREER Award CCR-9984314.

References

1. Bilas, A., Liao, C., Singh, J.P.: Using Network Interface Support to Avoid Asynchronous Protocol Processing in Shared Virtual Memory Systems. In *Proceedings of the 26th International Symposium on Computer Architecture*, May 1999.
2. Carter, J. B., et al.: Design of the Munin Distributed Shared Memory System. *Journal of Parallel and Distributed Computing*, **29**(2):219–227, September 1995.
3. Carter, J. B., et al.: Impulse: Building a Smarter Memory Controller. In *Proceedings of the Fifth International Symposium on High Performance Computer Architecture* January 1999.
4. Gokhale, M., Holmes, B., Iobst, K.: Processing in Memory: the Terasys Massively Parallel PIM Array. *Computer*, **28**(3):23–31, April 1995.

5. Hall, M., et al.: Mapping Irregular Applications to DIVA, A PIM-based Data-Intensive Architecture. *Supercomputing*, Portland, OR, Nov. 1999.
6. Heinrich, M., Speight, E.: Active Memory Clusters: Efficient Multiprocessing on Next-Generation Servers. Technical Report CSL-TR-2001-1014, Computer Systems Lab, Cornell University, August, 2001.
7. InfiniBand Architecture Specification, Volume 1.0, Release 1.0. InfiniBand Trade Association, October 24, 2000.
8. Keleher, P., et al.: TreadMarks: Distributed Shared Memory on Standard Workstations and Operating Systems. In *Proceedings of the Winter 1994 USENIX Conference*, pages 115–132, January 1994.
9. Kang, Y., et al.: FlexRAM: Toward an Advanced Intelligent Memory System. *International Conference on Computer Design*, October 1999.
10. Kim, D., Chaudhuri, M., Heinrich, M.: Leveraging Cache Coherence in Active Memory Systems. Technical Report CSL-TR-2001-1018, Computer Systems Laboratory, Cornell University, November 2001.
11. Kuskin, J., et al.: The Stanford FLASH Multiprocessor. In *Proceedings of the 21st International Symposium on Computer Architecture*, pages 302–313, April 1994.
12. Laudon, J., Lenoski, D.: The SGI Origin: A ccNUMA Highly Scalable Server. In *Proceedings of the 24th International Symposium on Computer Architecture*, pages 241–251, June 1997.
13. Lenoski, D., et al.: The Stanford DASH Multiprocessor. *IEEE Computer*, **25**(3):63–79, March 1992.
14. Li, K., Hudak, P.: Memory Coherence in Shared Virtual Memory Systems. In *ACM Transactions on Computer Systems*,7(4):321–359, November 1989.
15. Manohar, R., Heinrich, M.: A Case for Asynchronous Active Memories. In *ISCA 2000 Solving the Memory Wall Problem Workshop*, June 2000.
16. Message Passing Interface Forum. MPI: A Message-Passing Interface Standard, Version 1.0, 1994.
17. Nowatzyk, A., et al.: The S3.mp Scalable Shared Memory Multiprocessor. In *Proceedings of the 24th International Conference on Parallel Processing*, 1995.
18. Oskin, M., Chong, F. T., Sherwood, T.: Active Pages: A Computation Model for Intelligent Memory. In *Proceedings of the 25th International Symposium on Computer Architecture*, 1998.
19. Saulsbury, A., Pong, F., Nowatzyk, A.: Missing the Memory Wall: The Case for Processor/Memory Integration. In *Proceedings of the 23rd International Symposium on Computer Architecture*, pages 90–101, May 1996.
20. Scales, D. J., Gharachorloo, K., Thekkath, C. A.: Shasta: A Low-Overhead Software-Only Approach for Supporting Fine-Grain Shared Memory. In *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 174–185, October 1996.
21. Soundararajan, R., et al.: Flexible Use of Memory for Replication/Migration in Cache-Coherent DSM Multiprocessors. In *Proceedings of the 25th International Symposium on Computer Architecture*, pages 342–355, June 1998.
22. Speight, E., Bennett, J. K.: Brazos: A Third Generation DSM System. In *Proceedings of the First Usenix Windows NT Symposium*, August 1997.
23. Torrellas, J., Yang, L., Nguyen, A.-T.: Toward a Cost-Effective DSM Organization that Exploits Processor-Memory Integration In *Proceedings of the 6th International Symposium on High-Performance Computer Architecture*, January 2000.
24. Woo, S. C., et al.: The SPLASH-2 Programs: Characterization and Methodological Considerations. In *Proceedings of the 22nd International Symposium on Computer Architecture*, pages 24–36, June 1995.