A Multiprotocol Communication Support for the Global Address Space Programming Model on the IBM SP

Jarek Nieplocha Jialin Ju Tjerk P. Straatsma
Pacific Northwest National Laboratory, Richland, WA 99352, USA
http://www.emsl.pnl.gov/docs/parsoft/armci

Abstract. The paper describes an efficient communication support for the global address space programming model on the IBM SP, a commercial example of the SMP (symmetric multi-processor) clusters. Our approach integrates shared memory with active messages, threads and remote memory copy between nodes. The shared memory operations offer substantial performance improvement over LAPI, IBM one-sided communication library, within an SMP node. Based on the experiments with the SPLASH-2 LU benchmark and a molecular dynamics simulation, our multiprotocol support for the global address space is found to improve performance and scalability of applications. This approach could also be used in optimizing the MPI-2 one-sided communication on the SMP clusters.

1 Introduction

This work is motivated by applications that require support for a shared-memory programming style rather than just message passing. Many of them are characterized by irregular data structures, and dynamic or unpredictable data access patterns. For certain types of applications, the shared-memory programming model can be substituted or supported with the global address space model. Systems with a global address space usually do not offer coherent shared memory at the operating system level. Instead, in a distributed-memory environment they provide remote memory operations, for example as in the SHMEM library [1] on the Cray T3E, or one-sided communication operations in the MPI-2. The global address space can be used directly by applications, or indirectly supporting a shared-memory view of data structures emulated through a user-level library interface, such as the Global Arrays (GA) [2], that transparently to the user performs an appropriate translation of shared to distributed memory references. The programming model based on the global address space has the added benefit of preserving data locality information (the application is explicitly aware of, and controls data distribution) that well-written distributed-memory message-passing applications can exploit to increase performance. It also allows access the data in a fashion similar to shared memory without the interprocess synchronization imposed by the traditional cooperative message passing model.

In recent years, clustered systems with symmetric multi-processor (SMP) nodes have become increasingly popular as the cost effectiveness of SMP nodes and high-performance networks improved. An example of such an architecture is the IBM SP, a distributed-memory machine with SMP nodes, a network supporting high-performance user-space communication, and a rich programming environment that offers active messages and remote memory copy through the LAPI system library, threads, thread-safe MPI, and the standard Unix shared-memory interfaces within the

A. Bode et al. (Eds.): Euro-Par 2000, LNCS 1900, pp. 718-728, 2000.

SMP nodes. Our goal is to optimize communication support for the global address space on clustered SMP-based systems for applications that use one process/task per processor (rather than the explicitly multithreaded single process with per SMP node).

The main contribution of this paper is an integration of multiple communication protocols such as active messages, threads, and remote memory copy with shared memory to support the global address space model efficiently. Performance of one-sided operations is substantially improved by mapping the global address space to the shared-memory segments on SMP nodes. This technique makes a low-level messaging library such as LAPI responsible only for internode communication while the intranode communication is handled directly by shared memory. Performance advantages of this approach are shown for the remote-memory operations in the ARMCI portable communication library[3] and two applications, the SPLASH-2 LU benchmark and NWChem molecular dynamics code. The performance of NWChem was improved by 12.6% on 64 processors by relinking this large and already well tuned application unchanged with a different version of the ARMCI library. Although this paper focuses on the IBM SP, we used the multiprotocols for supporting global address space through ARMCI on other cluster platforms. We chose the SP since it 1) is a major cluster platform widely used for technical computing, 2) offers a richer set of vendor supported protocols (including active messages) relevant to our objectives than most other cluster platforms, and 3) all the discussed protocols are widely available in the user mode and supported on the current SP system configurations.

Shared memory has been exploited before in low-level one-sided messaging systems like Active Messages[4] or Nexus [5], and higher-level two-sided message-passing interfaces like MPI [6] on SMP clusters. However, as the programming models based on the global-address space and message passing are fundamentally distinct, different strategies are needed to pass benefits of shared memory to the applications. For example, the IBM implementation of MPI on SMP nodes exploits shared memory to move data between a pair of MPI tasks through an internal buffer in shared memory with one task copying data into the shared memory buffer and then the other copying into its separate address space. ARMCI places the application data in shared memory thus the data can be accessed without any intermediate memory copies and message buffer management overheads. It helps achieve 15 times better latency and 67% better bandwidth than in the vendor version of MPI within the SMP node. Similar performance gains are realized through the shared memory in ARMCI over LAPI within the node. On the IBM SP, we found that the shared memory optimizations benefit more the remote memory/one-sided operations than the message passing.

The rest of this paper is organized as follows: Section 2 describes integration of shared memory with LAPI and thread-based protocols; Section 3 reports the results of basic communication operations; Section 4 presents experimental results of two applications, the SPLASH-2 LU benchmark and the molecular dynamic simulation; Section 5 discusses related work. Finally, we conclude in Section 6.

2 SMP-Aware Communication Protocols

We developed ARMCI[3] to support remote memory operations in the context of distributed array libraries such as GA and compiler run-time systems such as the Parallel Runtime Consortium [7] Adlib. ARMCI supports remote memory copy, accumulate, and synchronization operations. It is portable and compatible with message-passing libraries such as MPI or PVM. Unlike most existing similar facilities,

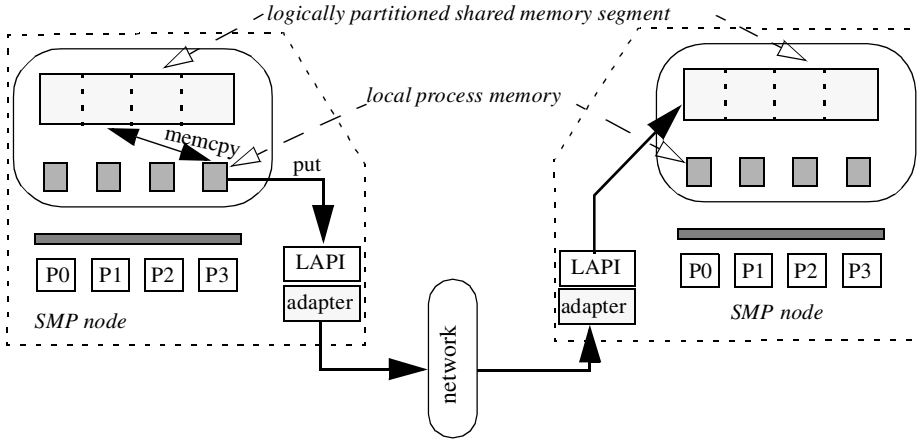

Figure 1: Combining shared memory with LAPI on the IBM SP

such as Cray SHMEM, it focuses on the noncontiguous data transfers that correspond to the data structures used in scientific applications (sections of multi-dimensional dense or sparse arrays, scatter, gather). Such transfers are optimized thanks to the non-contiguous data interfaces available for ARMCI data transfer operations: multi-strided and generalized UNIX I/O vector interfaces. ARMCI offers a simpler model and lower-level interface than the MPI-2 one-sided communication.

The standard parallel programming environment of the IBM SP includes LAPI, a low-level one-sided communication system that supports active messages (AM) and remote memory copy operations. LAPI offers competitive performance to MPI; however, unlike MPI it does not support noncontiguous data transfers which are common in scientific codes. ARMCI on the IBM SP uses active messages and threads rather than remote memory copies to optimize noncontiguous data transfers. Each process in a LAPI program uses at least three threads: the main application thread, and two extra threads for executing the LAPI active message handlers. In applications that use MPI in addition to LAPI, there are three additional threads introduced by the thread-safe IBM MPI library. For example, on the 4-way SMP node with 4 user processes (explicitly single threaded) that use MPI and LAPI, there are 4x6=24 threads. LAPI is supported in the SMP environment. However, since it uses the network adapter [8] to move data between processes on the same SMP node as if they were on separate nodes, its performance is not optimal. In particular, to transfer data between address spaces of two processes, LAPI performs at least two memory copies (to and from the adapter DMA area), and in many cases generates an interrupt.

We developed hybrid communication protocols for the ARMCI to exploit the SMP locality information and shared memory communication. The locality information is determined at start time from the process-to-node mapping, and used to select appropriate communication protocols. Within the SMP node, all operations are implemented on top of shared memory rather than with LAPI, see Figure 1. For inter-node communication, depending on the request size and shape, ARMCI chooses between active messages (AM) or remote memory copies to optimize bandwidth, see Figure 2. An implementation of the gather operation used by ARMCI is described in [9]. Since ARMCI includes a memory allocation routine to be used in the context of remote memory copy, memory is allocated using the System V shared memory

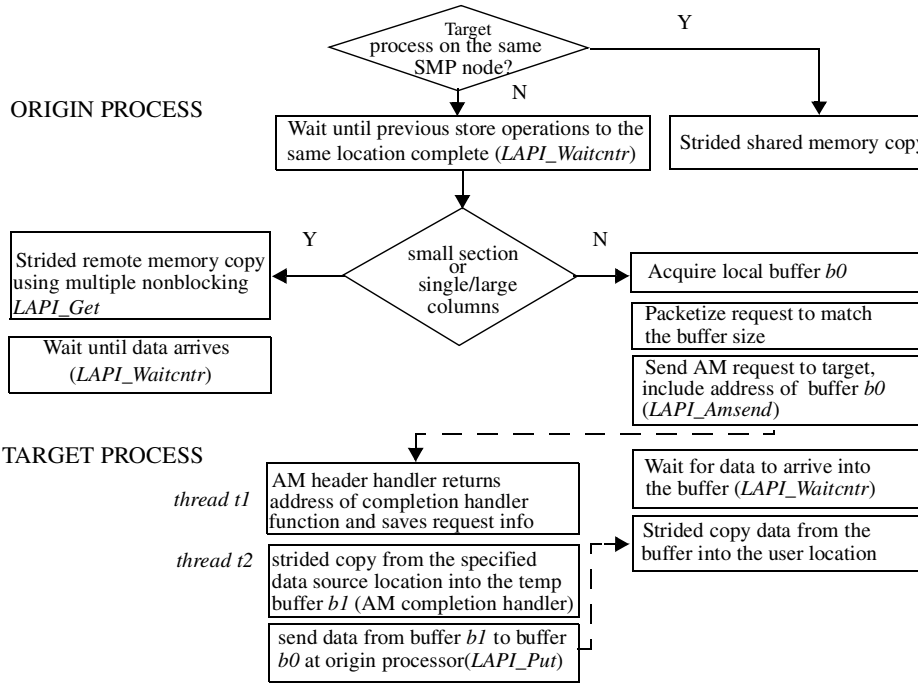

Figure 2: SMP-aware implementation of the ARMCI strided get operation on the IBM SP

interfaces. This critical function helps reduce the *put*/*get* operations to a memory copy and avoid at least one memory copy that LAPI must do. As explained below, the main difficulty is posed by the requirement to make multithreaded active-message internode protocols compatible with intranode shared memory operations.

ARMCI, in addition to remote memory copy, supports atomic operations: accumulate (reduction) as well as read-modify-write. The mutual exclusion embedded in the semantics of these operations requires special care when hybrid protocols are used. ARMCI on uniprocessor nodes uses active messages, threads, and Pthread mutexes. Since the same memory region in address space of process A can be addressed concurrently by a process B executing on the same SMP node and process C on a remote node, the mutual exclusion primitives must synchronize multiple threads in both the same and different process spaces. In AIX, Pthread mutexes cannot be used in this context. We designed a mutex lock replacement for both thread mutexes and System V semaphores. It offers a very low overhead and is free of the disadvantages of spin locks that can waste substantial amounts of CPU time by not yielding the processor when the mutex cannot be acquired for an extensive amount of time. We use the AIX atomic operation *check_lock* to check the content of a word. If the mutex is already acquired by another thread we use a spin lock with limited asymptotic backoff before finally yielding the processor to another thread.

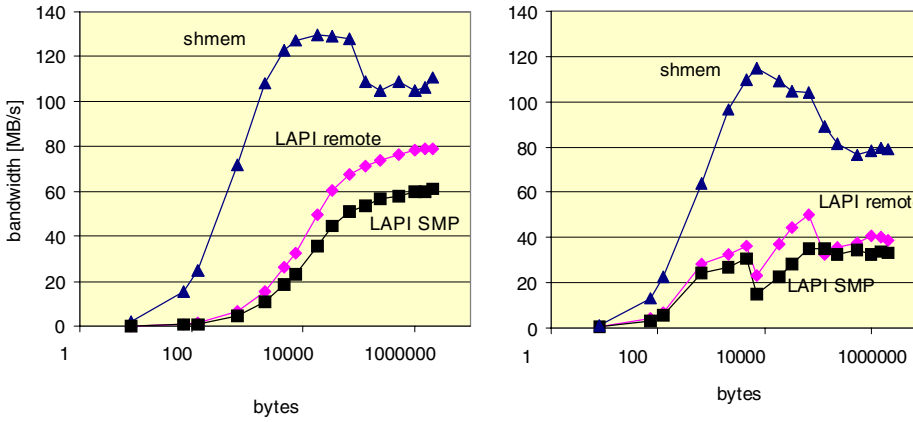

Figure 3: Performance of contiguous get (left) and contiguous accumulate (right) operations implemented using shared memory(shmem) and LAPI on the same SMP or remote node.

3 Performance of Communication Operations

We used a 16-node, 4-way SMP IBM SP with 64 PPC-604e processors and the TB3MX adapter at PNNL to study the performance of remote memory operations, the SPLASH-2 LU kernel benchmark and the molecular dynamics simulation. In this section, we discuss performance of the ARMCI get and accumulate operations in accessing contiguous and strided data on the same and remote SMP node. For the same node, the performance using the LAPI-based protocols and the shared-memory operations is presented.

Figures 3-4 demonstrate that our SMP-aware protocol outperforms LAPI by a large margin within a node. Interestingly, the observed bandwidth in the LAPI protocols used for intranode communication is in many cases worse than for the internode communication. This phenomenon is attributed to the fact that LAPI uses the network adapter even when communicating within the same node. For the intranode communication, the same adapter is used for sending and receiving the data, whereas for the internode communication the adapter handles only one side of the data transfer.

Despite the obvious differences between one-sided protocols in ARMCI and two-sided point-to-point message-passing protocols in MPI, we present performance of these systems to demonstrate how these interfaces take advantage of shared memory. Unlike LAPI, the IBM implementation of MPI already uses shared memory for communication within the SMP node. Table 1 shows latency and bandwidth numbers for the ARMCI get and the MPI send/receive operations. In this test, we used contiguous data transfers and tried to assure that the data is not already in cache. The MPI latency below is reported as 1/2 of the roundtrip time between two tasks for 1-byte message size, and we measured bandwidth for 512KB data size. The primary reason ARMCI outperforms MPI by a wide factor within the SMP node is that ARMCI get operation reduces simply to a memory copy whereas the MPI protocol adds the cost of message queue management and requires two memory copies and cooperation of two tasks to move data through an MPI internal shared memory buffer. Of course,

these advantages apply to the intranode environment. For the internode communication ARMCI uses LAPI, and in this case MPI and LAPI performances are similar [10].

The SMP-aware protocols, by improving communication performance within an SMP node, expose one additional layer of memory hierarchy in the IBM SP and provide an additional optimization opportunity to the applications.

Table 1: Performance of MPI send/receive and ARMCI get on the SMP node

interface	latency [μs]	bandwidth [MB/s]
MPI	13.2	65.63
ARMCI	0.85	109.64

4 Application Study

We used the SPLASH-2 LU benchmark and the NWChem molecular dynamics code to study the implication of SMP-aware communication protocols on the application performance. Neither of these codes was developed to exploit performance characteristics of the SMP-based clustered systems, and in this respect, are representative of the majority of scientific parallel codes. Since both codes perform internode and intranode communication, it was not clear what degree of performance improvement should be expected.

4.1 LU SPLASH-2 Benchmark

The SPLASH-2 benchmark suite [11] is a set of parallel applications for use in the design and evaluation of shared-memory multiprocessing systems. The suite contains two types of codes: full applications and kernels. We chose the LU program, which is one of the kernel programs from SPLASH-2, to evaluate the performance of our approach. The LU program factors a dense matrix into the product of a lower triangular and an upper triangular matrix. The factorization uses blocking to exploit temporal locality w.r.t. individual submatrix elements [12]. Originally designed to run on shared memory systems, this benchmark can only be used on a single SMP node of the IBM SP. Some modifications were needed to use the global address space model.

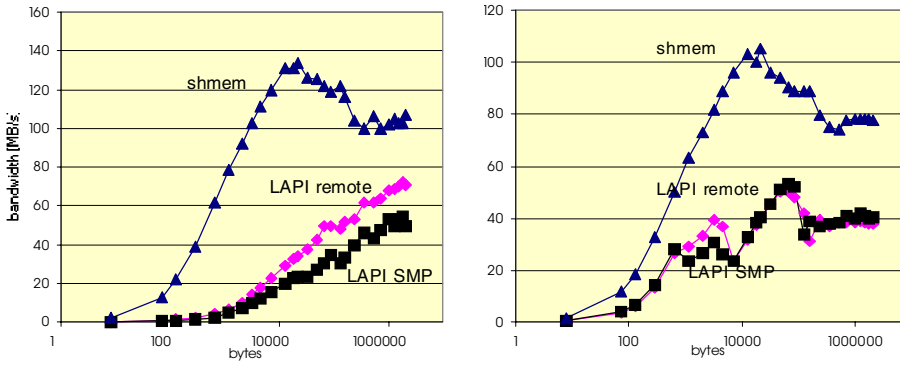

Figure 4: Performance of strided get (left) and strided accumulate (right) operations implemented using shared memory (shmem) and LAPI on the same SMP or remote node.

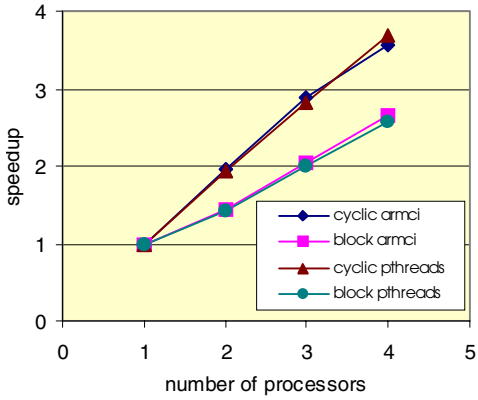

Figure 5: Speedup in the Pthread and ARMCI versions of the SPLASH-2 LU benchmark using block and block cyclic distributions on one SMP node

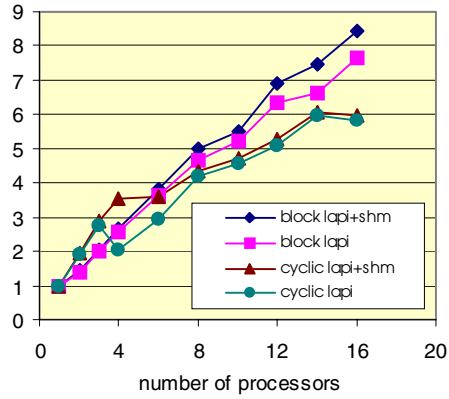

Figure 6: Speedup in the SPLASH-2 LU benchmark using block and block cyclic distributions and SMP-aware or SMP-oblivious communication protocols

We also developed a Pthread version of the benchmark to evaluate the performance of our modifications within an SMP node. The primary issues to be addressed included memory allocation, access to shared data, and interprocess/thread synchronization.

The Pthread version uses threads while the ARMCI version uses processes. In the first case, shared data is located in the process memory and accessed directly by the threads as needed. In order to replace shared memory with global address space, we had to divide the shared data, assign it to individual processes, and allocate the corresponding storage on each process. To synchronize processes, we used MPI_Barrier. Threads are synchronized with a Pthread mutex and a condition variable.

During the LU factorization, if required data blocks are in the local process memory, they are accessed directly. Otherwise, ARMCI_Get is used to copy the data block from the remote process that owns it to the local temporary storage. The computation requires transferring data blocks from the same row and column (for diagonal blocks). The original benchmark uses block cyclic distribution for load balancing. We also used block decomposition, as it had better locality of the data accesses. To take advantage of the SMP performance, the blocks are distributed according to a block pattern, such that the block that needs to be transferred has a better chance of residing in the local memory or neighboring memory on the same node. We used a matrix size of 3072 and a block size of 32 to study performance of the SPLASH-2 LU benchmark. The performance results given in Figure 5 indicate that the global address space and shared memory versions of the benchmark have similar performance for the same distribution types. The block cyclic distribution gives better performance than the block distribution because of better load balancing properties.

As the Pthread version works only on a single SMP node of the SP, we used only the ARMCI version of the benchmark on multiple nodes to study performance of the SMP-aware communication. We relinked the program with two alternative versions of ARMCI - one that uses only LAPI and the other that uses LAPI for internode communication and shared memory on a node. Since the block cyclic decomposition yields better load balancing the code scales better on a single SMP node. However, the lower memory reference locality compared to block decomposition causes

communication patterns to be spread out across all the nodes. Consequently, the benchmark performance on multiple nodes is better with block decomposition. To exploit performance advantages of both decomposition schemes, we also developed a third version of the benchmark that uses block cyclic decomposition on a single SMP node and block decomposition on multiple SMP nodes. Its performance is a good as its component protocols in their optimal operational regimes, see Figure 6.

The impact of using faster communication within SMP nodes is more significant with the block rather than block cyclic decomposition because it better exploits data locality. Improved performance is observed for the approach using LAPI and shared memory, although the difference is small when few processors are used. This is because more time is spent in computation than in communication. With more processors, the multi-protocol approach shows better performance, proving the benefits of shared memory support for the global address space.

Figure 6 shows some performance degradation effects in the LAPI-only version with four processors. We contribute this phenomenon to the hardware and software interaction involving the heavily used (for all the intranode communications) network adapter and scheduling of the 24 threads for a particular communication pattern in the benchmark. As our approach relieves the adapter from handling intranode communication any similar performance degradations have never been experienced.

4.2 Molecular Dynamics Simulation

NWChem is a massively parallel computational chemistry software package developed on top of Global Arrays (GA). This large (>600,000 code lines) package contains multiple methods for quantum-mechanical and classical computational chemistry including molecular dynamics. The recent version of GA employs ARMCI as its run-time system. GA maintains only the distributed array infrastructure, performs array index translation to the global address space, and implements collective array operations. All the GA one-sided communication is supported through ARMCI.

Molecular dynamics (MD) simulations typically evaluate atomic interactions within a specified cutoff distance. The implementation of the method in NWChem uses this locality of atomic interactions in the way the data is distributed on a distributed-memory MPP. This domain decomposition of the physical simulation volume exploits the locality of the atomic interactions such that all-to-all interprocessor communication is avoided. On computers where the cost of communication between processor pairs is not homogeneous, such as SMP clusters, an additional optimization of the domain decomposition is in principle possible for example by avoiding the assignment of physically distant parts of the simulation volume to processors capable of fast communication. In practice, this means that the locality in the simulation space is reflected in the choice of processor assignment. The processors on a given SMP node should typically be assigned adjacent parts of the simulation volume, for which communication requirements are always high.

Molecular dynamics simulations of liquid water were carried out with three versions of the code, see Figure 7. The application uses numbers of processors that are powers of 2. The original version is based on the ARMCI library that uses only LAPI for communication between and within SMP nodes. The other versions use the SMP-aware ARMCI library, with and without an additional permutation of the process numbers (inside GA) to improve communication locality based on our analysis of the MD communication patterns. No modification of the application code was needed.

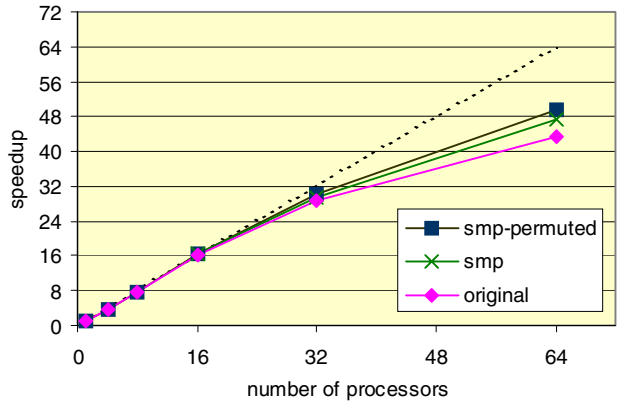

Figure 7: Performance of the MD simulation with SMP-aware and unaware (original) versions of ARMCI. A permutation of process numbers is used to improve communication locality

The three versions were produced by simply relinking the application with different versions of the ARMCI and GA libraries.

Similarly to the LU benchmark, the shared memory multiprotocols improved performance and scaling of this application. The improvement rate depends on the number of processors used and ratio of communication to calculation. The SMP optimizations are more effective when communication is made through a permutation of the process numbers. On 64 processors a 12.6% performance improvement is achieved over the original version. Since this well optimized application had already scaled almost linearly up to 32 CPUs, this is a substantial improvement, and it was achieved without any explicit modifications to this complex code. Moreover, 3-dimensional MD simulations on 4-way SMP clusters could not take full advantage of the communication locality. Our analysis indicates that with the increased number of CPUs per SMP node, the performance improvements should be even better.

5 Related Work

Multiprotocols involving shared memory has been used before to optimize performance of two- [6] and one-sided messaging systems [4,5]. Husbands and Hoe [6] used the shared memory mechanism for intra-SMP communication in an MPI implementation for a cluster interconnected by the StarT-X network. They optimized contiguous data transfer through a shared memory transfer facility in the MPICH channel layer on the node. ARMCI unlike MPI-1 supports one-sided communication. Lumetta *el.*, [4] proposed multi-protocol Active Messages on SMP cluster. Thir multi-protocol implementation of Active Message directs message traffic through the appropriate medium, either shared memory or network. Separate message queues are maintained for these two media. The shared memory queue block must first be mapped into address space of processes on the node. Message polling operations are ubiquitous in Active Message layers even with shared memory implementation, whereas ARMCI does not require polling. Also unlike [4], ARMCI incorporates threads among other protocols used. Foster, *el.*, [5] discussed the multi-protocol communication in the Nexus multi-threaded one-sided communication system that extends to heterogeneous platforms. Two-level protocols are available: one better suited for small, latency-

sensitive communications, and the other for large communications. The application performance can be optimized by using one link for synchronization and the other for data transfer. In ARMCI, selecting the protocol is not an issue anymore. Nexus unlike ARMCI does not include active messages among the protocols considered in [5].

The other differences between these papers and our work arise from the fact that ARMCI: 1) supports one-sided remote memory operations rather than one- or two-sided message-passing interfaces; 2) emphasizes noncontiguous data transfers (not addressed in papers [4-6]); and 3) supports atomic remote memory operations. In our experience, one-sided messaging libraries such as Nexus or Active Messages are very well suited for implementing inter-node communication but they add unnecessary overhead on the SMP node (e.g., related to message queue management, flow control, buffering) that can be avoided in the context of global address space model by using shared memory directly.

6 Conclusions and Future Work

We described a multiprotocol communication support for the global address space on the IBM SP that integrates shared memory within SMP nodes with the LAPI active messages, threads and remote memory copy between nodes. Shared memory offers substantial performance improvements over LAPI within a node for both contiguous and noncontiguous data transfers. Based on the SPLASH-2 LU benchmark and molecular dynamics simulation, the multiprotocol support for global address space is found to improve both performance and scalability of these applications. In the application context we have also found that 1) distribution plays an important role in exploiting the shared memory effectively and 2) replacing a shared memory programmming style (Pthreads) with the global address space model does not lead to performance losses within the SMP domain and provides good scaling across the cluster. Another important benefit of this approach is that the intranode communication is no longer involving the network adapter and its resources thus they can now be devoted exclusively to inter-node communications. This technique can also be used in implementations of the MPI-2 (not yet available on the IBM SP) on SMP clusters since it also offers a memory allocation interface (MPI_Alloc_mem) in the context of the one-sided operations. The described integrated protocols are employed in ARMCI for supporting the global address space model on more SMP-based systems than just the IBM SP. We intend to extend the ARMCI capabilities (e.g., support for heterogenous systems) and use it to implement other interfaces. For example, ARMCI is well suited for a portable implementation of the SHMEM library and we are also considering using it to support the MPI-2 1-sided “passive communication model”.

References

1. R. Barriuso, Allan Knies, SHMEM User’s Guide, Cray Research, SN-2516, 1994.
2. J. Nieplocha, R. J. Harrison, R.J. Littlefield. Global Arrays: A shared memory programming model for distributed memory computers. Proc. Supercomputing’94.
3. J. Nieplocha, B.Carpenter, ARMCI: A Portable Remote Memory Copy Library for Distributed Array Libraries and Compiler Run-time Systems, Proc. RTSPP IPPS/SDP’99, 1999.
4. S. Lumetta, A. M. Mainwaring, D.E. Culler, “Multi-Protocol Active Messages on a Cluster of SMP’s”. *Proc. Supercomputing’97*, 1997.

5. I. Foster, J. Geisler, C. Kesselman, S. Tuecke, "Managing Multiple Communication Methods in High-Performance Networked Computing Systems," *Journal of Parallel and Distributed Computing,* Vol. 40, January 1997.
6. P. Husbands, J.C. Hoe, "MPI-StarT: Delivering Network Performance to Numerical Applications". *Proc. Supercomputing'98*, 1998
7. Parallel Compiler Runtime Consortium, Common Runtime Support for High-Performance Parallel Languages, Proc. Supercomputing'93, 1993.
8. R. Govindaraju, IBM Power Parallel Systems, personal communication, 1999.
9. S.Andersson,G.Bhanot, J.Hague, F.Johnston, S. Kandalai, D. Klepacki, J. Levesque, J. Nieplocha, F. O'Connell, F. Parpia, C. Pospiech, *Scientific Applications in IBM RS/6000 SP Environments*, IBM Corp., ISBN: 0738415189, 1999.
10. M. Banikazemi, R. Govindraju, R. Blackmore, D. Panda, Implementing Efficient Implementation of MPI for IBM RS/6000 SP Systems, *Proc.* IPPS/SDP'99, 1999.
11. S.C. Woo, M.O. Ohara, E. Torrie, J.P. Singh, A. Gupta. "The SPLASH-2 Programs: Characterization and Methodological Considerations". *Proc. 22nd International Symposium on Computer Architecture*, 1995.
12. S.C. Woo, J.P. Singh, J.L. Hennessy, The Performance Advantages of Integrating Block Data Transfer in Cache-Coherent Multiprocessors. *Proc.6th ASPLOS*, 1994.