Exploiting Non-blocking Remote Memory Access Communication in Scientific Benchmarks

Vinod Tipparaju[1], Manojkumar Krishnan[1], Jarek Nieplocha[1],
Gopalakrishnan Santhanaraman[2], Dhabaleswar Panda[2]

[1] Pacific Northwest National Laboratory, Richland, WA 99352, USA
{Vinod.tipparaju, manojkumar.krishnan, Jarek.nieplocha}@pnl.gov

[2] Ohio State University, Columbus, OH 43210, USA
{santhana, panda}@cis.ohio-state.edu

Abstract. This paper describes a comparative performance study of MPI and Remote Memory Access (RMA) communication models in context of four scientific benchmarks: NAS MG, NAS CG, SUMMA matrix multiplication, and Lennard Jones molecular dynamics on clusters with the Myrinet network. It is shown that RMA communication delivers a consistent performance advantage over MPI. In some cases an improvement as much as 50% was achieved. Benefits of using non-blocking RMA for overlapping computation and communication are discussed.

1 Introduction

In the last decade message passing has become the predominant programming model for scientific applications. The current paper attempts to answer the question to what degree performance of well tuned application benchmarks coded in MPI can be improved by using another related programming model, remote memory access (RMA) communication. In the past RMA programming model was popular on the Cray T3D/E system where it was offered through the Cray SHMEM library [1]. The global address space architecture of these two Cray systems supported RMA communication very well. In fact, several of the current MPP systems only now can compete with latency and bandwidth of the RMA operations on the Cray T3E. In this comparative study, we are focusing on commodity Linux clusters with Myrinet. We chose this platform not because of its merits in supporting RMA but because of its popularity. In fact, Myrinet offers good support for message passing but rather limited support for RMA communication-- only the put operation has a native implementation in hardware. However, the next version of the Myricom GM programming interface will support get operation.

We use several popular scientific benchmarks and applications such as NAS CG and MG, SUMMA matrix multiplication, and Lennard Jones molecular dynamics to evaluate the effectiveness of RMA communication. In each case, two versions of the benchmark were derived: one based on blocking and the other non-blocking communication. The goal was to determine what additional performance benefit non-

blocking RMA can offer in each individual benchmark. This paper demonstrates even on a network with limited support for RMA, this communication paradigm can offer consistent performance advantages over message passing. These results are quite encouraging especially since the network vendors are offering increasing level of support for RMA communication and the expectation that the MPI-2 1-sided implementations will eventually become more widespread (not yet offered by Myricom in their MPICH-GM library). Note that in this paper, we are not trying to evaluate the rather complex model of the MPI-2 one-sided operations or investigate its potential for high-performance implementation [2] on modern networks such as Myrinet. Instead, by using ARMCI, a low-level high-performance portable RMA library with simple progress model, published implementation approach and performance results for Myrinet [3,4], we study what benefits the RMA communication can offer in general. In addition, ARMCI could be used in MPI codes as a high-performance alternative interface to the MPI-2 one-sided operations.

This paper is organized as follows. Section 2 describes RMA communication and the Myrinet network. Section 3 describes the benchmarks used in the study and gives a synopsis of how they were converted to use RMA. Section 4 presents experimental results. Section 5 summarizes related work and the paper is concluded in Section 6.

2 Remote Memory Access Communication on Myrinet

Remote memory access operations offer support for an intermediate programming model between message passing and shared memory. This model combines some advantages of shared memory, such as direct access to shared/global data, and the message-passing model, namely the control over locality and data distribution. RMA is sometimes considered a form of message passing; however, an important difference over the MPI-1 message-passing model is that RMA does not require an explicit receive operation and thus offers increased asynchrony of data transfers. The availability of non-blocking RMA operations presents additional opportunities for overlapping data transfers and computations. Although prefetching and poststoring instructions are often supported by the shared memory hardware and are exploited by compilers to overlap computations with data movement, a scientific programmer on shared memory systems typically faces difficulties when attempting to manage explicitly overlapping of computations and communication due to the lack of precise APIs. Such explicit non-blocking APIs are present in the most RMA interfaces.

We have been developing a portable RMA interface called ARMCI [5]. It is a rather low-level interface primarily intended as a run-time system for other programming models [21] such as Global Arrays [6], Co-Array Fortran [7] or UPC [8] compilers, or portable SHMEM library [9]. However, we also use it as the RMA communication layer for the benchmarks studied in this work. For Fortran codes, appropriate wrappers were added to access the needed functionality.

In the last few years, Myrinet has become a primary network for building medium and large-scale clusters based on commodity processing nodes due to its good scalability and relatively moderate cost. GM is a low-level message-passing system for the Myrinet network [10]. The GM system includes a driver, the Myrinet-interface

control program, a network mapping program, and the GM API, library, and header files. GM features include 1) concurrent, protected, user-level access to the Myrinet interface; 2) reliable, ordered delivery of messages; 3) automatic mapping and route computation; 4) automatic recovery from transient network problems; 5) scalability to thousands of nodes; and 6) low host-CPU utilization. GM has certain limitations including the inability to send messages from or receive messages into non-DMA-able memory, and offers no support for gather or scatter operations. Moreover, memory registration operations in GM under Linux are quite expensive relative to other systems [3].

The implementation issues of an extensive set of RMA interfaces on Myrinet clusters like those offered by ARMCI have been described before [3,4]. As Myrinet GM 1.x offers only support for the put operation, other RMA operations, such as get, are implemented using a client-server approach and the GM put operation. Recently ARMCI has been expanded to support non-blocking RMA operations. Their implementations extend the original client-server architecture in a manner that reduces the host CPU involvement in the communication. This is important for applications that attempt to overlap communication with computation.

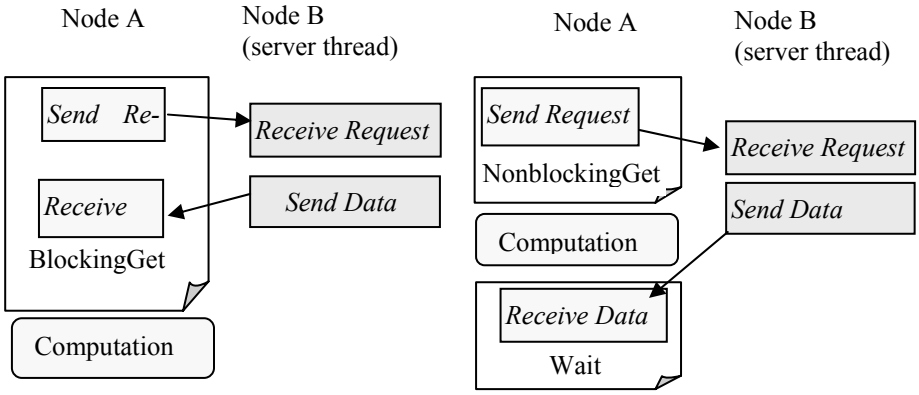


Fig. 1. Coupling of computation and communication in blocking (left) and non-blocking (right) get operation in ARMCI.

3 Application of RMA Communication in Scientific Benchmarks

To evaluate the benefits of RMA communication, we used multiple benchmarks representing a diverse set of algorithms used in scientific computing: conjugate gradient (CG) and multigrid (MG) kernel benchmarks from the NAS suite, SUMMA matrix multiplication, and a molecular dynamics application.

3.1 NAS Parallel Benchmarks

The NAS parallel benchmarks are a set of programs designed as a part of the NASA Numerical Aerodynamic Simulation (NAS) program originally to evaluate supercom-

puters. They mimic the computation and data movement characteristics of large-scale computations. NAS parallel benchmark suite consists of five kernels (EP, MG, FT, CG, IS) and three pseudo applications (LU, SP, BT) programs. Our starting point was NPB 2.3 [11] implementation written in MPI and distributed by NASA. We modified two of the five NAS kernels, MultiGrid (MG) and Conjugate Gradient (CG), to replace point-to-point blocking and non-blocking message-passing communication calls with first blocking and then non-blocking RMA communication. This is just a mere replacement of the point-to-point message passing communications part of the current message passing version of CG and MG NAS kernels using ARMCI RMA blocking and non-blocking operations. Other benchmarks (e.g., FFT, IS) rely on collective communication, thus limiting the appropriateness of RMA (point-to-point) communication with out reformulating the underlying mathematical algorithms. In our view, RMA is an alternative model to point-to-point message passing and a complementary model to collective operations. This view was shared by the authors of the Cray SHMEM library that offered both RMA and collective operations [1].

MG Benchmark

The NAS-MG multigrid benchmark solves Poisson's equation in 3D using a multigrid V-cycle. The multigrid benchmark carries out computation at a series of levels and each level of the V-cycle defines a grid at a successively coarser resolution. This implementation of MG from NAS is said to approximate the performance a typical user can expect for a portable parallel program on a distributed memory computer [11].

Most of the work in MG is done in four functions. Each of these functions is implemented using one or more 27-point stencils. “resid” is a function that computes the residual and operates at the same level of hierarchy. “psinv” is the smoother and also operates on the same levels of hierarchy. “interp” interpolates and “rpj3” projects between adjacent levels of hierarchy. The NPB 2.3 code uses a three-step dimensional exchange algorithm to satisfy boundary conditions. This is implemented with point-to-point message passing communication. In addition to this, point-to-point communication is used in the parallel implementation of these stencils to update every processors boundary values for each dimension that is distributed.

Our primary modification involved replacing these point-to-point communications with ARMCI RMA operations. For our implementation using ARMCI blocking operations, point-to-point communication was effectively replaced with the ARMCI_Put_notify operation. This blocking function call transfers the data to the destination processor memory and updates an internal (to the library) notify flag in the destination process memory. At the destination, arrival of this message can be (optionally) verified by making a call to ARMCI_Notify_wait that accesses the value of the notify flag. For our implementation using the corresponding non-blocking API, we attempt to achieve overlap by issuing non-blocking update in the next dimension before actually working on the data in the current dimension. This required us to use an additional set of buffers. Any explicit acknowledgement indicating the buffer availability is avoided by taking advantage of the periodic nature of the algorithm and alternating between these two sets of buffers.

CG Benchmark

In NAS CG benchmark, a conjugate gradient method is used to compute an approximation to the smallest eigenvalue of a large, sparse, symmetric positive definite matrix. This kernel benchmark tests irregular long distance communications and employs sparse matrix vector multiplication. The CG benchmark involves multiple iterations of a solution to the system of linear equations, Az=x, using the conjugate gradient method. It computes the residual norm at the end of each of these iterations. After each iteration, the eigenvalue is estimated with a shift λ. The size of the system, number of iterations involved and shift applied to the eigenvalue estimate is determined as a part of the initial setup of each class of the problem and is shown below.

Table 1. Problem sizes in the CG benchmark.

CLASS	N	Iterations	NonZeroes	λ
A	14000	15	11	20
B	75000	75	13	60
C	150000	75	15	110

Each of the CG iterations involves the following steps:

```
for i=1 to 25 {
  q = A.p        (A matrix vector product)
  α = ρ/(p^T q)  (dot product of result of the above
                  product and transpose of p)
  z  = z + α p
  ρ0 = ρ
  r = r − α q
   ….
}
Computation of the residual norm ||r||
```

Of the above, the steps that involve most of the communication are: matrix vector multiplication, vector dot product and computation of the residual norm. The computation of the matrix vector product involves a recursive doubling based pairwise exchange. This is implemented in the original algorithm using point-to-point communication. Since the recursive doubling based pairwise exchange is a barrier in itself, it is expected that replacing them with RMA operations would not give much benefit. Even the computation of the residual norm, which is a recursive reduction based pairwise exchange, synchronizes all the processes. Hence replacing the point-to-point communication with RMA blocking operations offers limited room for improvement. However using non-blocking RMA operations, we can overlap data exchange with sum of partial sub-matrices. This is done by overlapping communication and computation with in each exchange and between different exchanges by dividing the data into two parts and overlapping a communication operation involved in the exchange of one part of the data with the sum of partial sub-matrix vector product on the second part.

3.2 SUMMA – Matrix Multiplication

SUMMA is a highly efficient, scalable implementation of common matrix multiplication algorithm proposed by van de Geijn and Watts [12]. The MPI version is the SUMMA code developed by its authors, which is modified to use a more efficient matrix multiplication dgemm routines from BLAS rather than equivalent C code dis-

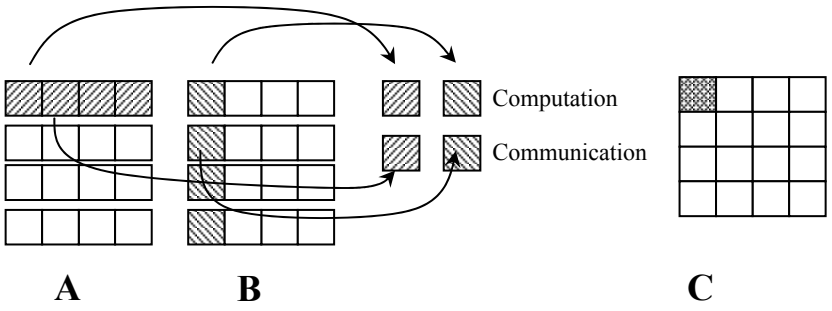


Fig 2. Using two sets of buffers to overlap communication and computation in matrix multiplication

tributed with SUMMA. We implemented two other SUMMA variants to use blocking and non-blocking RMA. The matrix is decomposed into sub-matrices and distributed among processors with a 2D block distribution. Each sub-matrix is divided into chunks. Overlapping is achieved by issuing a call to get a chunk of data while computing the previously received chunk, see Figure 2. The minimum chunk size was 128 for all runs, which was determined empirically and the maximum chunk size was determined dynamically, depending on memory availability and the number of processors.

3.3 Molecular Dynamics of Lennard-Jones System

Parallel molecular dynamics of a Lennard-Jones system is a benchmark problem that has been extensively used by various researchers [13-15]. Molecular dynamics (MD) is a computer simulation technique where the time evolution of a set of interacting atoms is followed by integrating their equations of motion. The force between two atoms is approximated by Lennard-Jones potential energy function U(r), where r is the distance between two atoms. Using Newton's laws of equation and Velocity-Verlet algorithm, the velocities and coordinates of the atoms are updated for the next time step. The physics of the molecular dynamics problem is described in [13].

$$U(r) = 4\varepsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right] \tag{1}$$

where, σ and ε are constants. The N atoms are simulated in a 3-D parallelepiped with periodic boundary conditions at the Lennard-Jones state point defined by the reduced

density $\rho = 0.8442$ and reduced temperature $T = 0.72$ [13]. The simulation is begun with the atoms on an fcc lattice with random velocities and with a time step of 0.004 in reduced units. For simplicity reasons, there are no neighbor lists or cutoff limits.

There are three main classes of parallelization for classical molecular dynamics: atom, force and spatial decomposition. In this paper, a parallel algorithm based on force decomposition is tested on a standard Lennard-Jones benchmark for problem size ranging from 256 – 100,000 atoms. There are three variants of the problem: message-passing, one-sided blocking and one-sided non-blocking RMA. The RMA versions were implemented using Global Arrays that manages distributed arrays and ARMCI for all communication. Force decomposition is based on a block decomposition of the force matrix F distributed among processors, where each processor computes a fixed subset of inter-atomic forces. The entire force matrix (N x N) is divided into multiple blocks (m x m), where m is the block size and N is the total number of atoms. Each process owns N/P atoms, where P is the total number of processors. Newton's third law is exploited as it halves the amount of computation.

In the MPI implementation, the force matrix owned by each processor P_z is of size $(N/\sqrt{P}) \times (N/\sqrt{P})$. As these elements are computed they will be accumulated into the corresponding force sub-vectors and finally folded together to get the total forces on its N/P atoms [13]. In the RMA implementation of Lennard-Jones, the force matrix and atom coordinates are stored in a global array. A centralized task list is maintained which stores the information of the next block that needs to be computed. The issue of load imbalance is a serious concern for force decomposition MD algorithm. Processors will have equal work only if the force matrix distribution is regular and equally sparse. In order to address load imbalance, a simple and effective dynamic load balancing technique called fixed-size chunking is used [16]. Initially, all the processes get a block from the task list. Whenever a process finishes computing its block, it gets the next available block from the task list. Overlapping of computation and communication is achieved by issuing a get call to the next available block in the task list, while computing a block.

Experimental Results

The experiments were performed on the 2.4GHz Pentium-4 Linux cluster with Myrinet-2000 at the State University of New York at Buffalo. It employs the most recent versions of GM and MPICH-GM libraries provided by Myricom.

We ran our MG tests for class A (problem size: 256X256X256, iterations: 4), B (problem size: 256X256X256, iterations: 20) and C (problem size: 512X512X512, iterations: 20). They are three production grade problem sizes for the MG benchmark.

For Class A, a smaller problem size with fewest iterations, ARMCI blocking code outperforms the reference MPI implementation by 7 to 30%. ARMCI non-blocking version achieves an additional overlap of 10 to 23% over the ARMCI blocking implementation and 28 to 46% improvement over the reference MPI implementation. Most of the overlap achieved over the blocking implementation is just by mere issue of the update in the next dimension while working on the current one. For Class B, with the same problem size as class A but more iterations, ARMCI blocking imple-

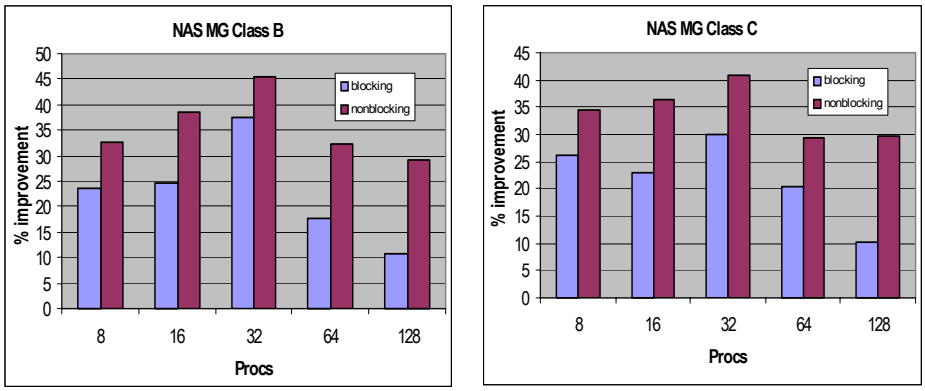


Fig. 3. Performance improvement in NAS MG for class B (left) and class C (right)

mentation outperforms MPI by 10 to 37%, see Figure 3 (left). ARMCI non-blocking implementation achieves an additional overlap of 5 to 20% over the blocking version and shows a 30 to 45% improvement over the reference MPI implementation. For Class C, ARMCI blocking implementation outperforms MPI by 10 to 32%. ARMCI non-blocking implementation achieves an additional overlap of 2 to 21% over the blocking implementation and shows a 30 to 40% improvement over MPI. Since coarser levels of multi-grid do not carry enough work to hide all the communication, an improvement achieved by using non-blocking over blocking API is limited for small processor configurations. With the increase of the number of processors for the problem size, the improvement is amplified.

Due to the synchronous nature of data transfers in the CG algorithm, the performance improvement over MPI, although consistent is rather limited, see Figure 4. As expected the main source of performance improvement is due to increased efficiency of RMA operations over the message passing (e.g., due to overheads associated with tag-matching, early message arrival etc that MPI must do). However, the non-blocking RMA offers an additional performance improvement. For example, for 128 processors it exceeds 10% over MPI.

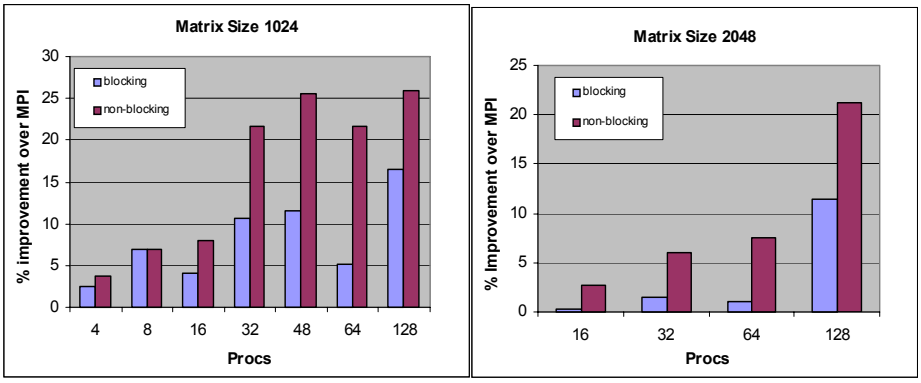


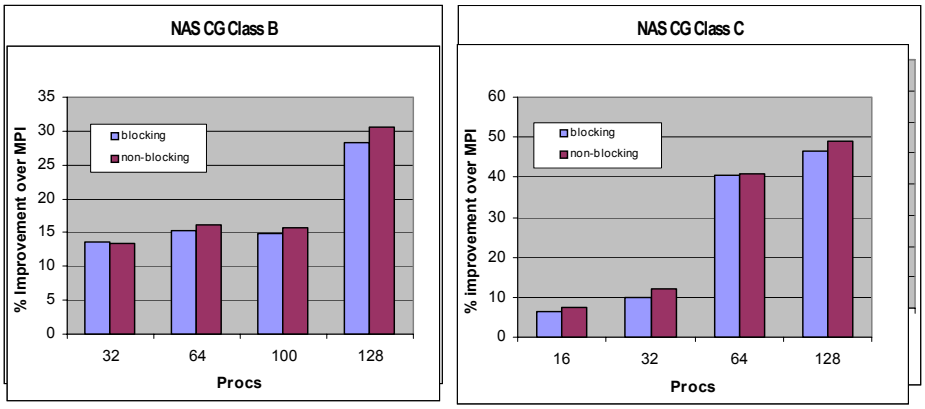
Fig. 5: Performance improvement in the matrix multiplication for matrix size 1024 (left) and 2048(right)

Experiments with matrix multiplication were run by varying the matrix size and the number of processors, with one and two processes per node. The results show that the RMA-based matrix multiplication consistently outperforms its message-passing counterpart by 10-25%. For a matrix size of 1024, as the number of processors increases, the amount of local computation is less and hence lesser overlap. On the other hand, for a large matrix size (e.g. 2048), initially the computation cost is very high when compared to the communication cost. As the number of processors increases the impact of communication cost comes into picture. The graphs in Figure 5 indicate that using RMA communication in SUMMA resulted in improved application performance over message passing. This performance benefit is mainly due to the efficiency of the communication layer (in this case, ARMCI), which reduced the data transfer cost when compared to message passing. However, for a very large problem size, the effect of overlapping computation is not perceived due to very high computation cost.

The experimental results of the molecular dynamics benchmark indicate that using RMA resulted in improved application performance over message passing, see Figure 6. This benchmark problem scales well when the number of processors and/or the problem size is increased, thus proving the solution is cost-optimal. In some cases, the performance improvement over MPI is greater than 40%. However, improvement in using non-blocking over blocking is not significant here as the potential for overlapping is limited in this benchmark problem.

5 Related Work

There have been multiple studies comparing effectiveness of different program-



] **Fig. 6**: Performance improvement in the molecular dynamics simulation involving 12000 (left) and 65536 (right) atoms.

ming models to MPI [17-20]. For example, paper [18] studies MPI, SHMEM and shared memory in the context of adaptive applications dynamic remeshing and the n-body problem, all on a shared-memory machine. Another related paper [19] is comparing different parallel languages to MPI in the context of NAS MG parallel benchmark. In both of these studies, MPI was hard to outperform. Despite important merits of the other models (ease of use, reduced implementation complexity) none of them showed a consistent performance advantage over MPI across all the discussed benchmarks. In [20] several benchmarks were used to compare performance of the KeLP C++ run-time library to MPI. By exploiting SMP locality and non-blocking communication in the KeLP data mover to overlap communication with computations performance, improvement from 12 to 28% was measured on a DEC cluster.

6 Conclusions

This paper compared performance of MPI and RMA implementations of four scientific benchmarks: NAS MG, NAS CG, SUMMA matrix multiplication, and Lennard Jones molecular dynamics on clusters with the Myrinet network. Both blocking and non-blocking RMA versions of the benchmarks were studied. In all these benchmarks, RMA delivered a consistent performance advantage over MPI. In some cases an improvement as much as 50% was achieved. In parts of the algorithms where overlapping communication with computations is possible, non-blocking RMA provided an additional performance boost. The overall performance advantage of RMA over the send/receive model can be contributed to the fact that this approach can avoid the overheads associated with typical implementations of MPI such as management of message queues, tag matching, and dealing with early arrival of messages. In addition, since explicit cooperation with the remote data owner is not needed for the data transfer to complete, RMA offers a more asynchronous programming model than MPI. However, this approach usually requires a careful program design to assure that the remote data is in consistent state when it is being accessed by the RMA calls.

Acknowledgments

This work was performed under the auspices of the U.S. Department of Energy (DOE) at Pacific Northwest National Laboratory (PNNL) and Ohio State University. PNNL is operated for DOE by Battelle. This work was supported by the Center for Programming Models for Scalable Parallel Computing project sponsored by the MICS/ASCR program in the DOE Office of Science.

References

1. R. Bariuso, Allan Knies, SHMEM's User's Guide,; Cray Research,, SN-2516, 1994.

2. Glen R. Luecke, Silvia Spanoyannis, Marina Kraeva, Comparing the Performance and Scalability of SHMEM and MPI-2 One-Sided Routines on a SGI Origin 2000 and a Cray T3E-600, J. PEMCS, December 2002.
3. J. Nieplocha, V. Tipparaju, A. Saify, D. Panda, Protocols and Strategies for Optimizing Remote Memory Operations on Clusters, Proc. Communication Architecture for Clusters Workshop of IPDPS'02. 2002.
4. J. Nieplocha, V. Tipparaju, J. Ju, E. Apra, One-sided communication on Myrinet, Cluster Computing, 6, 115-124, 2003.
5. 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.
6. J. Nieplocha, RJ Harrison, and RJ Littlefield, Global Arrays: A portable `shared-memory' programming model for distributed memory computers. Proc. Supercomputing'94, 1994.
7. R. Numrich, J.K. Reid, Co-Array Fortran for parallel programming. ACM Fortran Forum, 17(2):1-31, 1998.
8. W. W. Carlson, J. M. Draper, D. E. Culler, K. Yelick, E. Brooks, and K. Warren. Introduction to UPC and language specification. Tech Report CCS-TR-99-157, Center for Computing Sciences, 1999.
9. K. Parzyszek, J. Nieplocha and R. Kendall, A Generalized Portable SHMEM Library for High Performance Computing, Proc PDCS-2000, 2000.
10. Myricom, The GM Message Passing System, 10/16/1999.
11. D. Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, L. Dagum, R. Fatoohi, S. Fineberg, P. Frederickson, T. Lasinski, R. Schreiber, H. Simon, V. Venkatakrishnan, and S. Weeratunga, The NAS parallel benchmarks, Tech. Rep. RNR-94-007, NASA Ames Research Center, March 1994.
12. R. Van de Geijn and J. Watts. *SUMMA: Scalable Universal Matrix Multiplication Algorithm.* Concurrency: Practice and Experience, 9: 255-74, 1997.
13. S. J. Plimpton, "Fast Parallel Algorithms for Short-Range Molecular Dynamics", J. Comp. Phys.,117:1-19, 1995.
14. S. J. Plimpton. Scalable Parallel Molecular Dynamics on MIMD supercomputers. In Proceedings of Scalable High Performance Computing Conference-92, 1992.
15. K. Esselink, B. Smit, and P. A. J. Hilbers. Efficient Parallel Implementation of Molecular Dynamics on a Toroidal Network: I. Parallelizing strategy. J. Comp. Phys., 106:101-107, 1993.
16. C. Kruskal and A. Weiss. Allocating Independent Subtasks on Parallel Processors. IEEE Transactions on Software Engineering, vol. SE-11, no. 10, 1985.
17. Robert W. Numrich, John Reid, and Kieun Kim. *Writing a multigrid solver using Co-array Fortran*. In Proceedings of the Fourth International Workshop on Applied Parallel Computing, Umea, Sweden, June 1998.
18. H. Shan, J. P. Singh, R. Biswas, and L. Oliker. A Comparison of Three Programming Models for Adaptive Applications on the Origin2000. Proc. SC'2000.
19. Bradford L. Chamberlain, Steven J. Deitz, Lawrence Snyder, A Comparative Study of the NAS MG Benchmark across Parallel Languages and Architectures. SC'2000.
20. Scott B. Baden and Stephen J. Fink, Communication overlap in multi-tier parallel algorithms, Conf. Proc. SC '98, Orlando FL, November 1998
21. Center for Programming Models for Scalable Parallel Computing, www.pmodels.org.