Improving the Performance of Molecular Dynamics Simulations on Parallel Clusters†

Urban Borštnik, Milan Hodošček, and Dušanka Janežič*

National Institute of Chemistry, Hajdrihova 19, 1000 Ljubljana, Slovenia

Received November 11, 2003

In this article a procedure is derived to obtain a performance gain for molecular dynamics (MD) simulations on existing parallel clusters. Parallel clusters use a wide array of interconnection technologies to connect multiple processors together, often at different speeds, such as multiple processor computers and networking. It is demonstrated how to configure existing programs for MD simulations to efficiently handle collective communication on parallel clusters with processor interconnections of different speeds.

1. INTRODUCTION

Traditionally, supercomputers were used to meet the demands of numerically and computing-intensive applications in various fields of science. The concept of clustering workstations to form a supercomputer was present in research settings,[1] but as the performance difference between PCs and workstations disappeared, a cluster of PCs matching a supercomputer's performance became viable.[1,2] The original Beowulf cluster at NASA is composed of PCs connected by a single switch.[2]

A parallel program, in which processes running on different processors must communicate among themselves, loses computational time performing communication when waiting for data from other processors. An inherent advantage of supercomputers over networked PC clusters is their shared-memory design,[1] which allows fast access to data from other processes and enables programmers to take advantage of identical data being available to all processors. Since parallel clusters have distributed memory, they must use message-passing to achieve data exchange among processors.[1] For most programs that exchange a great deal of data between their processes, a supercomputer's faster data transport and lower latency is still an advantage over parallel clusters. New PC-class networking technologies, such as Myrinet[3] hardware and the Virtual Interface Architecture,[4] an efficient interface to the hardware,[5] provide higher speeds and lower latencies in networking to parallel clusters. The architecture of a parallel cluster must be taken into account when writing communication libraries, or alternatively, the network architecture may be designed to reflect a program's communication patterns.

To overcome these obstacles, a hierarchical hypercube parallel cluster topology was implemented.[6] A *d*-dimensional hypercube topology has 2^d processors, and every processor is directly connected to *d* other processors. If the processors are numbered in binary starting from 0, then a processor is connected to precisely the processors whose binary numbers differ only in one bit. For example, in a three-dimensional eight-processor hypercube, the third processor, 010, is connected to processors 110, 000, and 011.

Distributed memory is an inherent property of parallel clusters, necessitating a message-passing mechanism for data exchange. To aid in porting applications to different architectures and to provide an abstract view of complex communication operations, a message passing library is usually used in parallelized programs for data exchange among processors. It is this library that is optimized to run on different architectures—from supercomputers to parallel clusters. A message-passing library optimized to the underlying hardware may yield a significant increase in communication performance in a parallel cluster as well as in other parallel systems.[5,7−9]

There exists a trend toward increasing the number of processors in multiprocessor PCs using Shared Memory Multi Processors (SMP). Such PCs are effective for clusters, since the processors in a computer communicate among themselves using the local bus at speeds surpassing networking speeds. Considering that processors in different computers must communicate over a network, communication speed among them is usually lower. Also, computers are not necessarily connected with a network of the same speed. For example, Ethernet switches are cascaded into a tree structure when the number of computers in a cluster is very large.[10] The computers connected to two different switches must share the bandwidth available to connect the two switches. This bandwidth is generally lower than the bandwidth available to computers on the same switch. Another example of different network speeds is the use of Ethernet technologies offering different speeds, e.g., 100 Mb/s and 1 Gb/s, to connect computers in one cluster.[6]

Increasingly, parallel clusters are used for molecular dynamics (MD) simulations, a widely used tool in many fields of science.[11−13] MD integration methods can be implemented using different integration algorithms, which can be effectively parallelized because of the independent calculations of forces on atoms, which is the most demanding part of the simulation.[14,15]

In general, all widely used computer programs for MD simulations, e.g., CHARMM (Chemistry at HARvard Molecular Mechanics),[16] use an empirical potential function to accurately model the molecular system. The potential func-

* Corresponding author e-mail: dusa@cmm.ki.si.

† Dedicated to Dr. George W. A. Milne, a former long-term Editor-in-Chief of JCICS, for the opportunity to collaborate on a scientific and editorial level.

10.1021/ci034261e CCC: $27.50
Published on Web 02/13/2004

tion is a sum of many terms accounting for interactions among atoms. The CHARMM program[16] has been parallelized[17] to run on a variety of parallel computers, including parallel clusters with different logical topologies. A logical topology describes the pattern a parallel program uses to communicate among its processes, e.g., a parallel program whose processes communicate with only two neighbors has a ring logical topology.[18−20] A physical topology should be at least as well-connected as the logical topology of the parallel program.

In this paper we describe the performance gain of parallel MD simulations when processors are suitably placed in a logical topology. We also present a comparison of the MD simulation performance on various parallel clustering topologies.

2. METHODS

The parallel programs written for parallel clusters use a message-passing interface to exchange all data since the processors do not have a globally shared memory.

Two basic message-passing operations are *send* and *receive* that are used to transfer data between two processes; one process listens for and accepts data by calling a *receive* routine, while the other sends the data with a *send* routine. Even collective operations that transfer data among many processes, such as broadcasting from one processor to others or collecting data from others, are implemented mostly with individual *send* and *receive* routines called many times for different pairs of processes. A message-passing library contains a collection of basic and collective routines, allowing a programmer to focus on the data exchange in isolation from the details of its implementation or hardware used for communication. Libraries may have a variety of implementations of collective operations, allowing the most efficient implementation to be used according to the physical architecture on which a parallel program is run. Common message-passing libraries are based on the Parallel Virtual Machine (PVM)[21] and the Message Passing Interface (MPI) standards.[22] Different parallel programs for MD simulations may use different parallel approaches. The CHARMM program[16] provides its own MPI implementation, CMPI.[17] It includes specialized operations optimized for some topologies, most notably hypercubes; however, it still calls routines from an MPI library for basic *send* and *receive* operations.

An MD simulation consists of a number of time steps. Each step consists of the calculation of forces acting on atoms and the integration of the equations of motion to determine the new atom positions from the forces and the previous atom positions.[16] Both the force calculation and integration are parallelizible.[14,17] In the parallel version of CHARMM,[17] the force calculation is divided so that every process calculates forces for one part of a list of interacting atom pairs. For the parallel integration, the atoms are divided among processes, so that every process integrates equations for a subset of all of the atoms. An exchange of force data occurs among processes after the parallel force calculation and a broadcast of new atom position data to all processes occurs after the parallel integration. In the CHARMM program[17] as used for an MD simulation, the *vector distributed global sum* and *vector distributed global broadcast* operations are used for the exchange of force data and broadcast of new

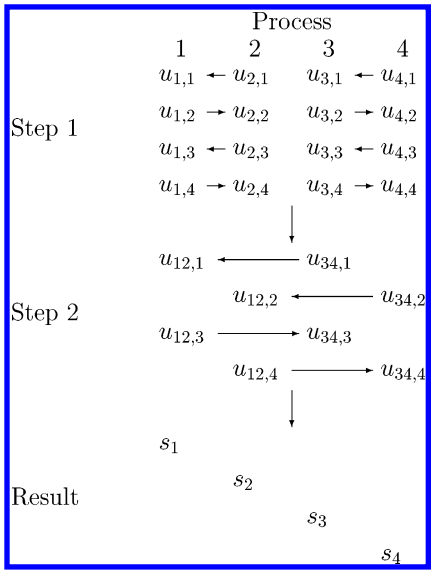

Figure 1. Depiction of hypercube-optimized *vector distributed global sum* with 4 processes. Each process n has a vector $\mathbf{v}_n = (u_{n,1}, u_{n,2}, u_{n,3}, u_{n,4})$, whose blocks $u_{n,1}$, $u_{n,2}$, $u_{n,3}$, $u_{n,4}$ contain the results of force calculations on the process. The result of the *vector distributed global sum* is the sum of vectors, $\mathbf{s} = \mathbf{v}_1 + \mathbf{v}_2 + \mathbf{v}_3 + \mathbf{v}_4$, and this sum $\mathbf{s}$ contains the total forces acting on atoms. The blocks of vector $\mathbf{s} = (\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3, \mathbf{s}_4)$ are left distributed among the processes: every processes has an appropriate block, e.g., process 1 has block $\mathbf{s}_1 = u_{1234,1} \equiv u_{1,1} + u_{2,1} + u_{3,1} + u_{4,1}$. Arrows indicate that the source is added to the destination, where the sum is left; e.g., in step 1, $u_{2,1}$ is added to $u_{1,1}$, resulting in $u_{12,1} \equiv u_{2,1} + u_{1,1}$ in place of $u_{1,1}$.

atom positions, respectively, and account for most of the interprocess communication. The vectors that these operations pass among the processes hold data on forces and atom positions, and their length is equal to the number of atoms. The vectors are logically divided into blocks, the number of blocks being equal to the number of processes. A correspondence exists between the processes and the blocks of every vector; i.e., every process is assigned a different block of atoms from a vector, but the division of the atoms into blocks is the same for all vectors.

In the parallel CHARMM program,[17] every step of the MD simulation begins with the force calculation. Each process calculates partial forces acting on atoms for a distinct subset of interacting atom pairs and stores the partial forces in a vector. The partial forces acting on each atom must then be added to obtain the total forces acting on the atoms, which are needed for the integration. Since processes perform integration for only a subset of all the atoms, the processes do not need the total force data for all of the atoms, but only for those for which they will calculate new positions. Hence the *vector distributed global sum* adds the vectors containing partial forces but leaves the blocks of the resulting total force vector distributed among processes. Every process has its corresponding block of the vector containing total forces acting on the atoms. The *vector distributed global sum* operation on four processes is illustrated in Figure 1. Every process now performs the integration of the equations of motion for its assigned block of atoms, obtaining their new positions, which are stored in the corresponding block of a vector. The *vector distributed global broadcast* then distributes the vectors containing new atom positions from every process to all of the processes, so that they all have

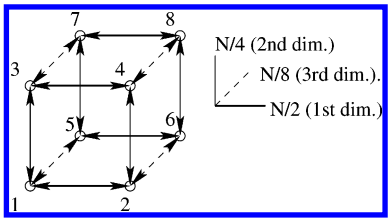

Figure 2. Amount of data transferred in a three-dimensional hypercube during a hypercube-optimized *vector distributed global sum* or *broadcast* operation as implemented in CMPI.

the complete set of new atom positions. The method by which the *vector distributed global broadcast* works is similar to the *vector distributed global sum* method in Figure 1, but the steps are performed in reverse order.

When CHARMM's CMPI implementation[17] is configured to run on a hypercube, it uses efficient algorithms for the *vector distributed global sum* and *broadcast* operations. It has been shown that both of these algorithms transmit and receive different amounts of data through different dimensions of a hypercube.[17,23] The amount of data transferred through each successive dimension is halved, which is illustrated in Figure 2. Both operations require one transfer, sequentially, through each dimension $d = \log_2 p$, where p is the number of processes. If N is the length of the vectors, then the most data, $N/2$, is transferred through the first dimension, half of that amount, $N/4$, is transferred through the second dimension, and similarly for higher dimensions. The least data, N/p, is transferred through the final dimension (i.e., the fifth dimension when there are 32 processes).[16]

If all of the connections between the processes are of equal speeds, then not all of the connections are used efficiently with this algorithm. Only connections representing the first dimension are saturated; therefore, these limit the performance of the whole topology. If the underused connections representing higher dimensions are replaced with slower ones, forming a hierarchical physical topology, then no significant performance is lost.

Assuming that one process runs on one processor, we can map and number the processes onto the processors and study the communication pattern among processors. By numbering the processors in binary starting from zero, then the processors differing in the *i*th bit are connected through the *i*th dimensions; e.g., processors 12 (1010) and 14 (1110) are connected through the third dimension. If we have a hierarchical topology, then the processors connected through lower dimensions have a higher or equal speed to processors connected through higher dimensions. Thus, the processes of the parallel program using the *vector distributed global sum* and *broadcast* should be correctly mapped to the physical topology, so the dimensions used by the program to communicate matches the dimensions of the physical topology. Then, the most data are transferred through the fastest connections, while slower connections are used to transfer data for higher dimensions.

When running CHARMM in parallel[16] on PC clusters, the user or batch queueing system must specify a host file listing computers on which processes are executed. If the number of processes is a power of two, then CMPI[17] uses efficient hypercube-aware collective communication routines that transfer a different amount of data along the different dimensions of the hypercube.[17] The ordering of computers in the host file determines where in the logical hypercube the process is placed. It is advantageous to place processes running on one SMP computer as neighbors on the lowest dimensions of the hypercube, since they exchange more data.

Table 1. Optimal Computer Ordering for the Host Files Used by the MPI Libraries for SMP PCs[a]

number of processors	computer order in host file
1 × 2	1 1
2 × 2	1 2 1 2
4 × 2	1 3 2 4 1 3 2 4
8 × 2	1 5 3 7 2 6 4 8 1 5 3 7 2 6 4 8
16 × 2	1 9 5 13 3 11 7 15 2 10 6 14 4 12 8 16
	1 9 5 13 3 11 7 15 2 10 6 14 4 12 8 16
2 × 4	1 2 1 2 1 2 1 2
4 × 4	1 3 2 4 1 3 2 4 1 3 2 4 1 3 2 4
8 × 4	1 5 3 7 2 6 4 8 1 5 3 7 2 6 4 8
	1 5 3 7 2 6 4 8 1 5 3 7 2 6 4 8

[a] A 8 × 2 number of processors means eight dual-processor computers for a total of 16 processors.

The host file is used by a MPI library to execute processes on specified computers.[24] An algorithm, based on bit reversing, that produces an appropriate host file to be used for CHARMM reads as follows:

Input:

d dimension of hypercube = log_2 of the number of processors

smp number of processors per computer

offset lowest-numbered computer in list

Output:

a host file listing computers

print(*number_to_name*(*offset*), " 0")

for *line* := 1 **to** 2^d **do**

reverse := *reverse_bits*(*line*)

computer := *offset* + ⌊*reverse*/*smp*⌋

print(*number_to_name*(*computer*), " 1 MDProgram")

end

The *reverse_bits* function returns the parameter in reverse bit order. For example, the number 12 (1010 in binary) would be returned as 5 (0101 in binary). The *number_to_name* function returns a locally visible computer host name given a computer serial number, usually adding a prefix and suffix string; for example, a value of 3 would return *node3.cluster.site.int*. The first line specifies that no additional process should be executed on computer n1, on which the first process of the program is started manually or by the batch queueing system. The remaining lines specify the program to be run on the appropriate computers.

The crucial aspect of the host file is the computer ordering. The optimum order for a number of SMP PC configurations is listed in Table 1.

In examining the performance of the parallel programs, an important study is to examine the scaling properties of the programs. As the number of processors on which a program runs is increased, the parallel programs tend to run less efficiently because the time lost to communication time increases. The time lost to communication is reflected in the

Table 2. Overall Speedups and the Time Spent (in s) Only for Communication for Three Different Network Setups and Three Different Parallel Libraries[a]

	CMPI[b]									
CPUs	1000[c]		100 + 1000[c]		100[c]		MPICH[b] 1000[c]		LAM/MPI[b] 1000[c]	
1 × 2	1.88	5.1	1.89	5.2	1.89	4.9	1.89	6.4	1.89	4.9
2 × 1	1.97	5.7	1.96	5.7	1.89	24.6	1.96	6.6	1.96	7.2
2 × 2	3.63	7.8	3.64	7.9	3.45	21.7	3.59	12.9	3.58	13.3
4 × 1	3.78	7.5	3.66	14.7	3.40	35.4	3.74	9.4	3.65	17.2
4 × 2	6.74	10.0	6.48	16.6	5.85	32.9	6.40	17.8	6.45	17.8
8 × 1	7.11	8.6	6.52	20.8	5.76	40.5	6.94	11.5	6.59	19.2
8 × 2	12.02	11.2	10.48	23.1	8.67	42.3	10.53	22.8	10.31	25.4
16 × 1	12.79	9.5	10.57	25.7	8.74	45.1	12.04	14.4	10.50	26.2
16 × 2	19.98	12.3	15.40	26.5	11.53	48.0	15.50	26.0	13.13	37.2

[a] The model system is an HIV Integrase enzyme and surrounding water consisting of 54212 atoms using a 19/20 Å force switch cutoff. A CPUs of 4 × 1 means only one CPU is used in each of 4 PCs, while 4 × 2 means two CPUs are used in 4 PCs, for a total of 8 CPUs. The 1000 network is a 1 Gb/s Ethernet switch, the 100 + 1000 network is a 100 Mb/s Ethernet switch with 1 Gb/s Ethernet point-to-point links, and the 100 network is a 100 Mb/s Ethernet switch. [b] Library. [c] Network.

speedup, i.e., how much faster a program runs on N processors than on one:

$$\text{speedup} = \frac{\text{run time on 1 processor}}{\text{run time on } N \text{ processors}}$$

An ideal parallel program would have a linear speedup, meaning that the speedup is equal to the number of processors. In this case, doubling the number of processors would halve the running time. Most programs have speedups that are lower than the ideal.

3. RESULTS

A comparison was made of several parallel network architectures and software libraries, running the parallel CHARMM program,[16] and to verify the impact of properly ordering the host list that specifies computers on which to execute processes.

The CROW8 (Columns and Rows of Workstations)[6] cluster features 60 dual-processor computers connected to three cascaded gigabit switches. The computers have dual AMD MP-2200+ processors running at 1.8 GHz, and 512 Mb of RAM. The gigabit Ethernet cards are Intel 82540EM-based, and the switches are Netgear GS524T 24-port switches. The C library on the computers is glibc 2.3.1, and the C and FORTRAN compilers were gcc and g77, versions 3.2.2. The Linux kernel version was 2.4.20. A LevelOne FSW-2409TFX 24-port switch was used for some of the tests.

For the tests, three different network architectures were used. With a gigabit switch (1000 network), all computers communicated using the gigabit Ethernet switch. With a fast Ethernet switch (100 network), all the computers communicated using the fast Ethernet switch. In the hierarchical hypercube (100 + 1000 network), all of the computers were connected to the fast Ethernet switch but were also linked in pairs using gigabit point-to-point links, as shown in Figure 3. In this case, the links of the cluster, onto which the dimensions of the hierarchical hypercube are mapped, have different speeds. The first dimension is mapped to the fastest link, which is the local bus between two processors in each PC, reaching speeds of 2.6 Gb/s. The second dimension is mapped to the point-to-point 1 Gb/s Ethernet links between pairs of PCs. The third through fifth dimensions are mapped to the 100 Mb/s Ethernet switch. Additionally, the use of only one processor was compared to using both processors in the PCs.

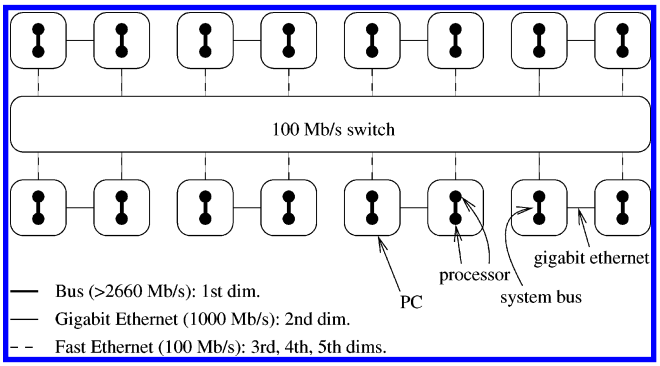

Figure 3. CROW8 topology. The local bus between processors (thick lines) is used for the first dimension of a hypercube, the gigabit links between PCs (thin line) is used for the second dimension, while the fast Ethernet links (dashed line) between PCs and the switch are used for the third and higher dimensions. The links are progressively slower with higher dimensions.

To monitor the speedup of the parallel programs on the cluster, short simulation runs of a molecular system were used as a benchmark. The model system was an HIV Integrase enzyme[25] with surrounding water molecules, consisting of 54 212 atoms. The 500-step MD simulations of this system with a 1 fs step size and a 19/20 Å force switch cutoff was performed by the CHARMM[16] version c29a2. The presented speedups are obtained from the best running time of several simulation runs.

A performance comparison was made of two CHARMM versions compiled with three MPI libraries: CMPI,[17] MPICH,[26] and LAM/MPI.[27,28] One version of CHARMM used CMPI, CHARMM's internal implementation of collective operations with MPICH[26] for simpler communication. The other two versions of CHARMM used only the MPICH[26] and LAM/MPI[27,28] libraries for all communication, including collective operations. In these cases, CHARMM uses the MPI_REDUCE_SCATTER and MPI_ALLGATHERV subroutines from the MPI libraries for the *vector distributed global sum* and *broadcast*, respectively. As shown in Table 2, the version of CHARMM using CMPI performed better than the versions of CHARMM using either MPICH or LAM/MPI on the gigabit switch.

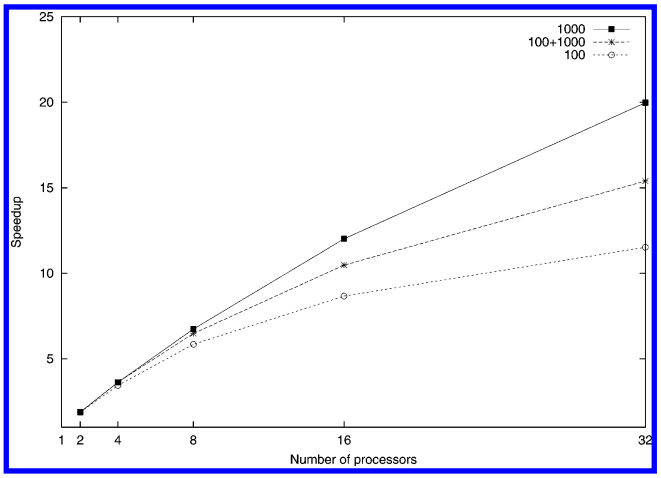

Figure 4. Speedups of different network architectures: 1 Gb/s switch, a 100 Mb/s switch with 1 Gb/s point-to-point links, and a 100 Mb/s switch using the CMPI communications library in CHARMM.

The scaling of the CHARMM program[16] was also examined when using different networks: the 1000, 100 + 1000, and 100 network, using the CMPI library. As the results from Table 2 and the graph in Figure 4 indicate, the 1000 network yields the best performance and the 100 network yields the lowest performance. Adding 1 Gb/s point-to-point links to the 100 Mb/s switch to form the hierarchical hypercube (100 + 1000 network) noticeably improves performance as compared to the 100 network. The faster point-to-point links are used for data transfer of the lower dimensions, which provides the speed increase visible for any number of processors greater than four. However, the 100 + 1000 network does not reach the performance of the 1000 network for greater numbers of processors. The performance of the 100 + 1000 network matches the performance of the 1000 network when up to 4 processors are used, since the 100 Mb/s switch is not used in these cases and only the local bus and 1 Gb/s networking technologies are used. As the number of processors increases, the 100 Mb/s Ethernet switch is used for data transfer of a greater number of higher dimensions. Even though less data is transferred for these, higher, dimensions than the lower ones which use the local bus and faster networking, the use of a slower switch for higher dimensions still leads to a degradation in the performance of parallel computation.

A comparison of the performance when using only one processor versus two processors in the computers reveals that using two processors yields a better performance than using one processor per computer. The only case in which using two processors performed better was with the 100 network. Since this network is much slower than the other networks, especially the local bus between processors, it is advantageous to use both processors, thereby limiting the amount of data transferred over the slower network. In all of the other cases, the overhead of using two processors for computation was greater than the gain provided by faster networks. Table 2 also lists the times spent for communication. For the 1000 network and all three libraries, the communication times for the same total number of processors was greater when using two processors per computer than when using only one processor per computer and twice as many computers. When using the slower 100 + 1000 and 100 networks, the communication times were lower when using two processors.

4. CONCLUSIONS

The present work provides the procedure to obtain a performance gain for MD simulations on parallel clusters. We have compared the scaling of the parallel CHARMM program using three different libraries, three different network architectures, and using one and two processors per computer. The results indicate that a hierarchical topology, meaning that different processors are connected by different speeds, can bring a noticeable improvement in the performance of the MD simulation programs with an appropriate message-passing library aware of the topology. The results are of considerable interest, particularly due to the importance of large-scale simulations to study complex molecular systems.

ACKNOWLEDGMENT

The financial support of the Ministry of Education, Science and Sports of Slovenia under grant number P01-0104-503 is gratefully acknowledged.

REFERENCES AND NOTES

(1) Spector, D. H. M. *Building Linux Clusters: Scaling Linux for Scientific and Enterprise Applications*; O'Reilly & Associates: Sebastopol, CA, 2000.

(2) Sterling, T.; Becker, D. J.; Savarese, D. Beowulf: A parallel workstation for scientific computation. In *Proceedings, International Conference on Parallel Processing*, 1995.

(3) Boden, N. J.; Cohen, D.; Felderman, R. E.; Kulawik, A. E.; Seitz, C. L.; Seizovic, J. N.; Su, W.-K. Myrinet: A gigabit-per-second local area network. *IEEE Micro* **1995**, *15(1)*, 29−36.

(4) *Virtual Interface Architecture Specification Version 1.0*; Technical report; Compaq and Microsoft and Intel: December 1997.

(5) Buonadonna, P.; Geweke, A.; Culler, D. An implementation and analysis of the virtual interface architecture. In *Proceedings of the 1998 ACM/IEEE conference on Supercomputing*. *IEEE Computer Society*. **1998**, 1−15.

(6) Hodošček, M.; Borštnik, U.; Janežič, D. CROW for large scale macromolecular simulations. *Cell. Mol. Biol. Lett.* **2002**, *7(1)*, 118−119.

(7) Sistare, S.; vande Vaart, R.; Loh, E. Optimzation of MPI collectives on clusters of large-scale SMP's. In *Proceedings of the ACM/IEEE SC99 Conference* 1999.

(8) Shan, H.; Singh, J. P.; Oliker, L.; Biswas, R. Message passing and shared address space parallelism on an SMP cluster. *Parallel Computing* **2003**, *29(2)*, 167−186.

(9) Karonis, N. T.; de Supinski, B.; Foster, I.; Gropp, W.; Lusk, E.; Lacour, S. *A Multilevel Approach to Topology-Aware Collective Operations in Computational Grids*; April 2002.

(10) Dietz, H. G.; Mattox, T. I. KLAT2's flat neighborhood network. In *Extreme Linux track of the 4th Annual Linux Showcase;* October 2000.

(11) Janežič, D.; Praprotnik, M. Symplectic molecular dynamics integration using normal mode analysis. *Int. J. Quantum Chem.* **2001**, *84*, 2−12.

(12) Grubmüller, H.; Heller, H.; Windermuth, A.; Schulten, K. Generalized verlet algorithm for efficient molecular dynamics simulations with long-range interactions. *Mol. Sim.* **1991**, *6*, 121−142.

(13) Janežič, D.; Praprotnik, M. Molecular dynamics integration time step dependence of the split integration symplectic method on system density. *J. Chem. Inf. Comput. Sci.* **2003**, *43(6)*.

(14) Heermann, D. W.; Burkitt, A. N. *Parallel Algorithms in Computational Science*; Springer-Verlag: Berlin, 1991.

(15) Murty, R.; Okunbor, D. Efficient parallel algorithms for molecular dynamics simulations. *Parallel Computing* **March 1999**, *25(3)*, 217−230.

(16) Brooks, B. R.; Bruccoleri, R. E.; Olafson, B. D.; States, D. J.; Swaminathan, S.; Karplus, M. CHARMM: A program for macromolecular energy, minimizationj, and dynamics calculations. *J. Comput. Chem.* **1983**, *4(2)*, 187−217.

(17) Brooks, B. R.; Hodošček, M. Parallelization of CHARMm for MIMD machines. *Chem. Design Autom. News* **1992**, *7*, 16−22.
(18) Trobec, R.; Jerebic, I.; Janežič, D. Parallel algorithm for molecular dynamics integration. *Parallel Computing* **1993**, *19(9)*, 1029−1039.
(19) Trobec, R.; Šterk, M.; Praprotnik, M.; Janežič, D. Implementation and evaluation of MPI-based parallel MD program. *Int. J. Quantum Chem.* **2001**, *84(1)*, 23−31.
(20) Trobec, R.; Šterk, M.; Praprotnik, M.; Janežič, D. Parallel programming library for molecular dynamics simulations. *Int. J. Quantum Chem.* **2004**, *95*, 530−536.
(21) Sunderam, V. S. A framework for parallel distributed computing. *Concurrency: Pract. Experience* **1990**, *2(4)*, 315−339.
(22) Snir, M.; Otto, S.; Huss-Lederman, S.; Walker, D.; Dongarra, J. *MPI: The Complete Reference*; MIT Press: Cambridge, MA, 1996.
(23) van de Geijn, R. A. *LAPACK working note 29 on global combine operations*; Technical Report CS-91-129; University of Tennessee: April 1991.
(24) Gropp, W. D.; Lusk, E. *User's Guide for mpich. A Portable Implementation of MPI*; ANL-96/6; Mathematics and Computer Science Division, Argonne National Laboratory: 1996.
(25) Berman, H. M.; Westbrook, J.; Feng, Z.; Gilliland, G.; Bhat, T. N.; Weissig, H.; Shindyalov, I. N.; Bourne, P. E. The protein data bank. *Nucl. Acids Res.* **2000**, *28*, 235−242. http://www.pdb.org/.
(26) Gropp, W.; Lusk, E.; Doss, N.; Skjellum, A. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing* **1996**, *22(6)*, 789.
(27) Burns, G.; Daoud, R.; Vaigl, J. LAM: An Open Cluster Environment for MPI. In *Proceedings of Supercomputing Symposium* 1994; pp 379−386.
(28) Squyres, J. M.; Lumsdaine, A. A Component Architecture for LAM/MPI. In *Proceedings, 10th European PVM/MPI Users' Group Meeting*, number 2840 in Lecture Notes in Computer Science, Venice, Italy September/October 2003, Springer-Verlag.

CI034261E