Parallel Molecular Dynamics Simulations of Alkane/Hydroxylated α-Aluminum Oxide Interfaces [1]

S. Roy [b], R.Y. Jin [a], V. Chaudhary [b] and W. L. Hase [a]

[a] *Department of Chemistry*

[b] *Department of Electrical and Computer Engineering*

Wayne State University, Detroit, MI 48202

Abstract

In this paper we describe a practical implementation of parallel computation for the molecular dynamics (MD) simulation of an alkane/aluminum oxide interface. A serial MD program was converted into a parallel code utilizing the message passing interface (MPI). This code was evaluated on a twelve processor symmetrical multiprocessor as well as on a cluster of four processor SMPs. A maximum speedup of 5.25 was achieved with twelve processors on the large shared memory machine. The cluster performance saturated at a speedup of 4.5 with two nodes. High communication costs and considerable load imbalance in the system were identified as areas that need further investigation for obtaining better performance.

Key words: Molecular dynamics, message passing, SMP cluster.

1 Introduction

Molecular dynamics (MD) studies are widely used to investigate the structural and dynamical properties of a wide range of materials. The popularity of MD is mainly due to its ability to provide a detailed description of the motions and interactions of every particle in a system. The microscopic insight obtained from MD is particularly valuable for understanding dynamical processes at the atomic level.

[1] This work was supported in part by grants from Ford Motor Company, the Office of Naval Research, and National Science Foundation

Molecular dynamics simulations are inherently computationally intensive due to a detailed force evaluation at each iteration step and the small iteration time step. The number of force centers in the system and conditions of the simulation such as temperature, often limit the size of the molecular system and length of the trajectories that can be practically studied with scalar computers.

Parallel and distributed processing is a promising solution to future computing requirements. Significant advances in parallel algorithms and architecture have demonstrated the potential for applying concurrent computation techniques to a wide variety of problems [1–5,7]. In this paper, we present a practical implementation of parallel computing techniques performing a molecular dynamics simulation study of adhesive interactions between alkanes and a hydroxylated aluminum oxide surface.

2 Parallel Computation

The principal idea of parallel computing is that the computational load is divided and distributed to more than one processor so that the overall computation time is reduced. When applied to molecular dynamics simulations, the amount of computer time a processor spends on a serial code can be partitioned as follows:

$$T_{seq} = T_p + T_s$$

T_p represents CPU time on the force evaluation which consists of evaluation of the bonded and nonbonded energy and forces and generation of the nonbonded neighbor list. T_s represents operations that can not be made parallel.

In a parallel environment with n processors, assuming perfect load balance, computer time spent on the ideal parallel code would be reduced to:

$$T_n = T_p/n + T_s$$

Addition of more processors into the parallel computing system decreases the CPU time on each node progressively. For example, in a typical MD application, where T_p accounts to more than 90 % of CPU time, the parallel program with a large number of processors should ultimately achieve a speedup of:

$$
\begin{aligned}
\text{Speedup} &= \frac{T_{seq}}{T_n} \\
&= \frac{T_p + T_s}{T_p/n + T_s} \\
&= \frac{1}{0.9/n + 0.1} = 10 \ (n \to \infty)
\end{aligned}
$$

Although it seems that a large number of processors should result in a sizable reduction in CPU time, the overall performance of the parallel application is never quite realized in practice due to communication overhead that accompanies parallel computing. This overhead consists of latency time, ie. time required to establish a communication link between two processors, and cost of transferring the necessary data between the processors. Communication overhead is usually insignificant for a relatively small number of processors, since the cost of computation for each node is larger than the cost for communication. However, as the number of processors increases, the communication cost becomes an important consideration relative to the decreasing computational load.

3 Molecular Dynamics

The code used here is a molecular dynamics program developed in house for the study of liquid alkane films confined between hydroxylated aluminum oxide surfaces. The details of the code have been previously discussed [18] and will only be briefly summarized here. Periodic boundary conditions are imposed in both the x and y directions. The z direction is perpendicular to the hydroxylated aluminum oxide surface. The force centers are of the united atom (UA) type. The potential energy function consists of intramolecular bonded interactions represented by internal coordinate bond distance, valence angle, and dihedral angle harmonic terms and nonbonded interactions expressed in terms of inter-atomic distances. The simulations are carried our under conditions of fixed number of particles, constant volume and constant temperature (NVT) [19]. Temperature control is achieved by Berendsen's method [20]. The Velocity Verlet algorithm [21] was used to integrate the classical equations of motion.

3.1 Partition of Molecular Interactions

There are two general approaches to decomposition and distribution of the computational load among the processors [3, 7–17].

3.1.1 Domain Decomposition

In domain decomposition [8–10, 12, 13, 15, 16], the space of the molecular system is divided into subspaces, the force centers within the subspace are then assigned to a processor. The main difficulty in implementing domain decomposition is due to connectivity of the molecules extending from one subspace to another. In addition, computing pairwise interaction also requires frequent communication update of positions of force centers in neighboring subspaces. Therefore, additional computation must be introduced in the domain decomposition scheme. As a result, the domain decomposition strategy is generally reserved for very large molecular systems.

3.1.2 Force Decomposition

The second strategy, which is employed here, is commonly referred as particle or force decomposition. The force centers in the molecular system are divided by certain partition schemes to achieve an equal distribution of computational load on each processor [3, 13, 14]. Unlike domain decomposition, in which the responsibility of each processor lies in its own subspace, in particle decomposition, each processor is responsible for a number of force centers within the entire space. As a result, each processor must possess identical copies of information such as atomic positions and forces for the entire molecular system. The results from each processor at the end of each time step are communicated and incorporated into a global data type before the program proceeds to the next iteration. The amount of communication is relatively large compared to that in domain decomposition. Thus, though the particle decomposition strategy is easier to implement, the size of the molecular system that can be practically studied is usually limited by memory requirements and communication overhead. Hence, the particle decomposition method typically works well for relatively small molecular systems.

Recent developments in computer hardware however has greatly expanded the size of the molecular system where the particle decomposition strategy can be effectively utilized. The chosen alkane – Al_2O_3 system presented here 2000 UA $CH_2(CH_3)$ groups and 4406 force centers within the hydroxylated Al_2O_3 solid surfaces.

4 Simulation Procedures

Molecular dynamics simulations are carried out by solving the classical equations of motion for the dynamics of molecules according to a potential energy force field. Newton's equation of motions are numerically integrated to predict the destination of every atom in the molecule within a short time interval. A time-dependent trajectory of molecular motions is then constructed for all the atoms by using successive time steps, and the structural and dynamics properties are computed as time averages. This requires that forces exerted on each atom should be calculated at every time step. As described above, in our alkane – Al_2O_3 system, the force evaluation consist of both bonded and nonbonded interactions.

4.1 Bonded Force Calculations

For bonded forces, the alkane chains are treated as a sequence of oscillating spherically symmetric atoms linked by covalent bonds, modeled as simple harmonic springs. The interactions of adjacent atoms involve bond stretching described by a simple harmonic potential energy function,

$$U_s = \frac{1}{2}k_s(r - r_0)^2 \tag{1}$$

where k_s, r, and r_0 are the bond stretching force constant, bond length, and equilibrium bond distance respectively.

The bond angle between two connecting bonds is also constrained. The bending of this angle is defined by a harmonic interaction potential energy function,

$$U_\theta = \frac{1}{2}k_\theta(\theta - \theta_0)^2 \tag{2}$$

where k_θ, θ, and θ_0 are the bond angle bending force constant, bond angle and equilibrium bond angle respectively.

In addition to the bond stretching and angle bending, a four-centered torsional interaction potential function is also introduced,

$$U_\phi = \frac{1}{2}k_n\left[1 + \cos n(\phi - \phi_0)\right] \tag{3}$$

where k_n is the torsional force constant that gives the strength of the restraint, n is an integer periodicity or number of symmetry, ϕ and ϕ_0 represent the torsional angle and equilibrium torsional angle respectively.

Since bonded interactions only involve up to 3 neighbors along the chain, partition of bonded interactions can be achieved simply by assigning equal number of force centers to each participating processor.

4.2 Nonbonded Force Calculation

The nonbonded interactions consist of an intramolecular component which involves atoms separated by more than three successive bonds within the same chain, as well as intermolecular forces between atoms from neighboring molecules. The nonbonded force U_{nb} between the atoms i and j is represented by the Lennard-Jones potential,

$$U_{nb}(r_{ij}) = 4\varepsilon \left[\left(\frac{\sigma}{r_{ij}} \right)^{12} - \left(\frac{\sigma}{r_{ij}} \right)^{6} \right] \tag{4}$$

where r_{ij} is the distance between atom i and atom j. ε and σ represent the nonbonded interaction parameters.

Although the computational load involved with the evaluations of the bonded interactions grows linearly with the size of the system, the energy expression for the nonbonded interactions exhibits a quadratic dependence on the number of atoms. Hence, the evaluation of the nonbonded interactions is generally the most computationally intensive constituent of the MD code.

A commonly used approximating technique in the nonbonded force calculation, to increase the simulation speed, is a nonbonded cutoff. Since the Lennard-Jones potential decays quickly to zero at large molecular separation, it is possible to ignore the atoms which are far away and whose contribution to the force is insignificant. Therefore, for each atom, a neighbor search is performed throughout the space, and only the neighboring atoms falling within the cutoff sphere are included for force evaluation. The significance of the nonbonded cutoff is that the computational dependency now scales linearly with the number of atoms. Nevertheless, the nonbonded force evaluation remains one of the most computationally intensive part of MD simulations.

4.3 Decomposition of Nonbonded Interactions in the Bulk

In order to compute the nonbonded forces, the intermolecular distance has to be computed. The introduction of the nonbonded cutoff requires that the force centers within the cutoff radius must be known. This is achieved by the construction of a nonbonded neighbor list. The list of neighboring atoms is maintained, and updated at a certain time interval for each atom. For the nonbonded forces within the bulk molecular system, the list is an $N \times N$ matrix, where N represents the number of atoms. Observing Newton's third law, an important observation can be made that for each force exerted by atom i on atom j, atom j exerts an equal but opposite force on atom i.

Therefore, the interaction matrix is symmetric and the computational load can be further conveniently cut in half. With elimination of half of symmetric entries, the neighbor list matrix now takes the shape of a triangle [3].

The partition of the computation load can no longer be linear since the load distribution must now account for the fact that more work is involved for the atoms in the upper part of the triangle than the bottom ones. It has been demonstrated that for a collection of P processors, equal partition of this triangular neighbor list can be efficiently achieved by dividing the nonbonded list into strips of equal areas [3].

$$i_p = N - \left\lfloor \sqrt{\frac{P-p}{P}} N \right\rfloor \tag{5}$$

where i_p represents the largest particle index assigned to processor p, P is the total number of processors, and N is number of force centers.

The number of atoms assigned to each processor is determined at the beginning of the program and remains the same throughout the simulation. It has been shown that this static data partition scheme works quite well in balancing the computational load among the processors [3].

4.4 Decomposition of the alkane – Al_2O_3 interfacial interactions

The partition of nonbonded interactions between the bulk alkanes and the hydroxylated aluminum oxide surface requires a different approach. The surface is treated as a rigid entity, and the forces are computed only for the atoms in the bulk. If the solid surface contains S force centers, the nonbonded neighbor list for the interactions between the bulk and the surfaces then is a $N \times S$ matrix. Since only the atoms near the solid surfaces are included in the nonbonded force computation, all the surface neighboring atoms entries in the matrix are absent except for those near the surface. Data partition is accomplished by assigning the atoms alternatively to the processors in a cyclic fashion.

4.5 Communications

It is essential for an efficient parallel MD application that all processors start a computational job simultaneously and terminate concurrently. The procedure for each processor is flowcharted in Figure 1.

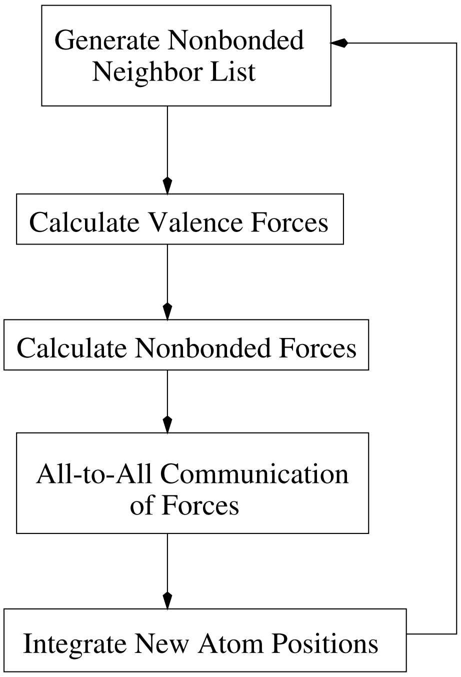

Fig. 1. Flowchart of MD simulation

In the procedure list, the first three steps can all be accomplished concurrently and independently for each processor. Since communication among all the processors cannot be completed unless the data from every node is available, the completion of the global all-to-all communication step will depend on the availability of data from the slowest processor, that is, the one with the most computational load. Hence, the successful implementation of parallel computation greatly depends on the assurance of equal distribution of computational load among all the participating processors.

5 Results and Discussion

A parallel implementation of a serial MD code was evaluated on a cluster of SUN UltraEnterprise multi-processor workstations. The alkane $-Al_2O_3$ system simulated consists of 2000 UA $CH_2(CH_3)$ groups confined between two hydroxylated Al_2O_3 surfaces each consisting of 2203 force centers. The simulations were performed for 1000 iteration time steps.

The cluster consists of a SUN UltraEnterprise E4000 with six processors, and three SUN UltraEnterprise E3000s with four processors each. The E4000 can be loaded with up to twelve processors. The nodes in the cluster are connected with an 155 Mbps ATM network using a ForeSystems switch, and 100 Mbps FastEthernet using a BayNetworks hub.

Figure 2 shows the parallel speedup using the E4000 loaded with twelve processors (intrabox communication), a cluster of E3000s with twelve processors (three E3000s, each with four processors) using FastEthernet, and a cluster of E3000s with twelve processors (three E3000s, each with four processors) using ATM OC-3 network. A speedup of 3 is achieved with four processors on the E4000, and roughly 5 for eight processors. Although adding more processors seems to increase the performance further, the improvement appears marginal.

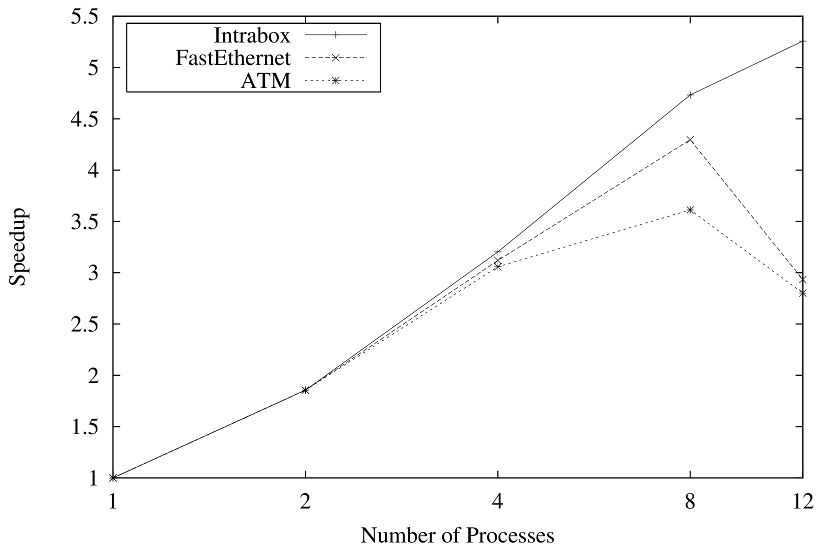

Fig. 2. Speedup using 12 processor E4000 (intrabox), 12 processor E3000 cluster connected by FastEthernet, and 12 processor E3000 cluster connected by ATM OC-3. The E3000 cluster has three E3000s, each with four processors.

For the cluster, there are four processors in each box, so that two boxes are required for an allocation of 8 processors. It can be seen that the performance degrades somewhat for 8 processors, and is severely compromised for 12 processors. As explained earlier, the parallel MD code uses global summing to distribute the updated force values across the processors at every time step. The overhead due to these reduction operations is shown as a percentage of total elapsed time in Figure 3.

It can be seen that the communication cost increases with addition of more processors. Although the computational load on each processor is decreased when adding more processors, this benefit can be compromised by the resulting increased communication overhead. When the increase in communication cost supersedes the savings in the computation time, further addition of processors will actually cause performance degradation in a parallel application. This is reflected in the case of 12 processors using FastEthernet and ATM in Figure 2.

A detailed examination of the communication behavior of the program shows that the first data exchange accounts for between 33 % – 49 % of the total overhead. This abnormally large component arises due to load imbalance

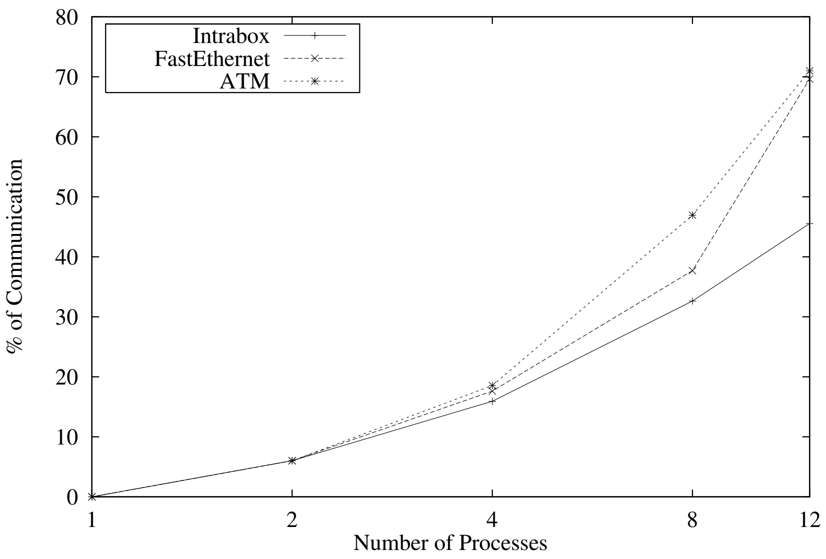

Fig. 3. Overhead due to reduction operations in the E4000 (12 CPUs), E3000 cluster with FastEthernet, and E3000 cluster with ATM OC-3. This includes the cost of communication and wait times due to load imbalance.

among the processors. All nodes are waiting for the slowest processor to arrive. Figure 4 shows the percentage overhead of load imbalance estimated from the time taken for the first global communication at each iteration.

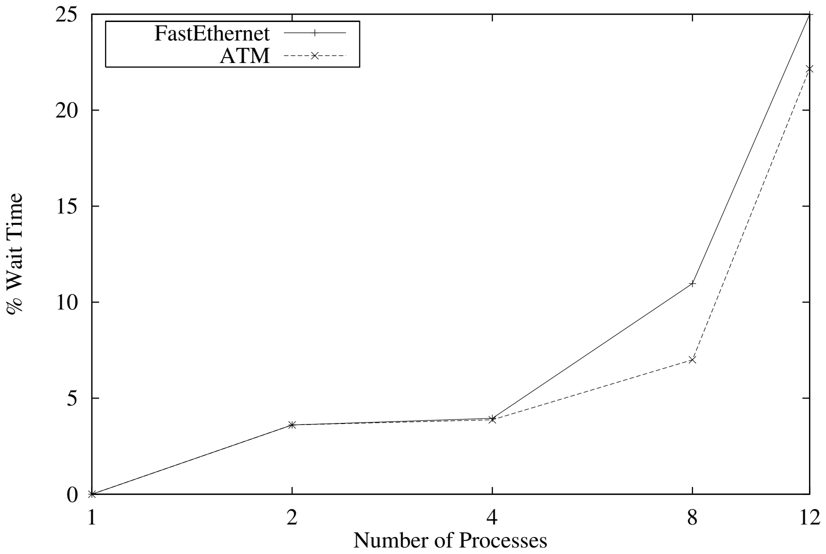

Fig. 4. Overhead due to load imbalance in the E3000 cluster connected by FastEthernet versus E3000 cluster connected by ATM OC-3. Estimated from the time taken for the first reduction operation per time step.

The load imbalance arises due to the unequal load distribution while calculating the nonbonded interactions. Figures 5, 6, and 7 show the "incremental" fraction of the total number of nonbonded interactions that each processor computes for the case of 4, 8, and 12 processors, respectively. The incremental fraction corresponds to the contribution of each processor towards the entire computation (which is up to a scale of 1.0). The distance between two lines indicates the fraction contributed by a processor. The contribution of the first processor is taken from the 0.0 line and that of the last processor is from the 1.0 line. It can be seen that the contribution due to each processor is not equal, as opposed to the results in [3].

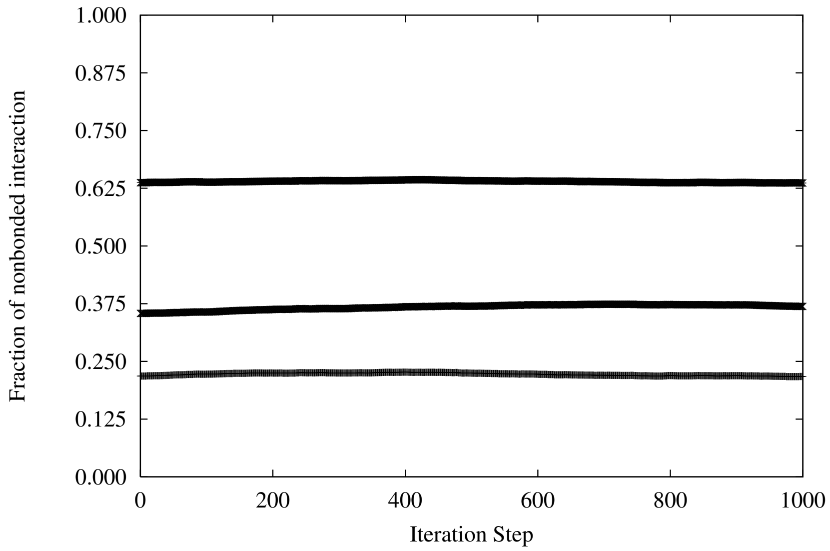

Fig. 5. Incremental fraction of nonbonded interactions calculated by each processor, with a total four processors. The y-axis indicates the total load, as a fraction with each line (of the graph) indicating the fraction of load due to addition of one processor (as we go up the y-axis).

6 Conclusion

This paper describes a molecular dynamics (MD) simulation study of an alkane/aluminum oxides system using a parallel program. This code was evaluated on a 12 processor SMP and was found to achieve a maximum speedup of approximately 5. The code was also evaluated on a cluster of SMPs. The cluster performance saturated at a speedup of 4.5 with two nodes and 8 processors. Though the overall speedup values compare favorably with the results obtained while studying bulk phenomena [3], high communication costs and considerable load imbalance were found in the system. The communication cost is inherent in using the force decomposition approach, but the load imbalance may need further investigation for obtaining better performance.

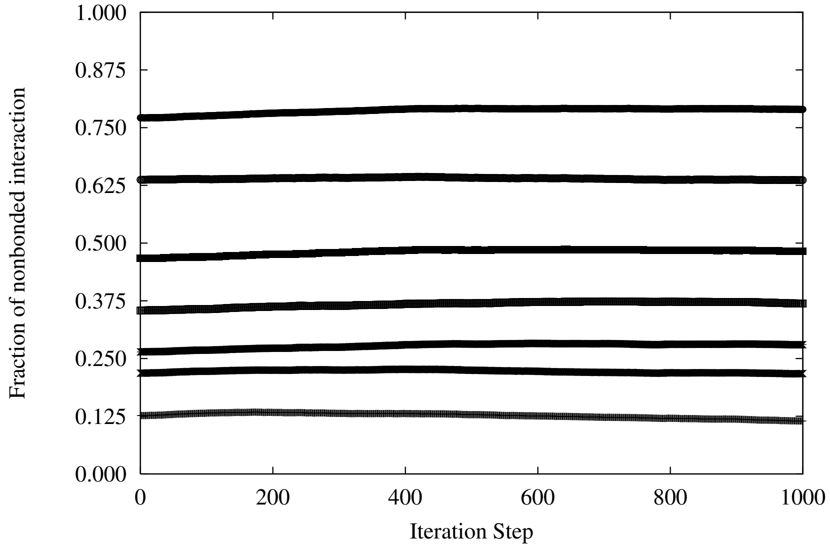

Fig. 6. Incremental fraction of nonbonded interactions calculated by each processor, with a total of eight processors. The y-axis indicates the total load, as a fraction with each line (of the graph) indicating the fraction of load due to addition of one processor (as we go up the y-axis).

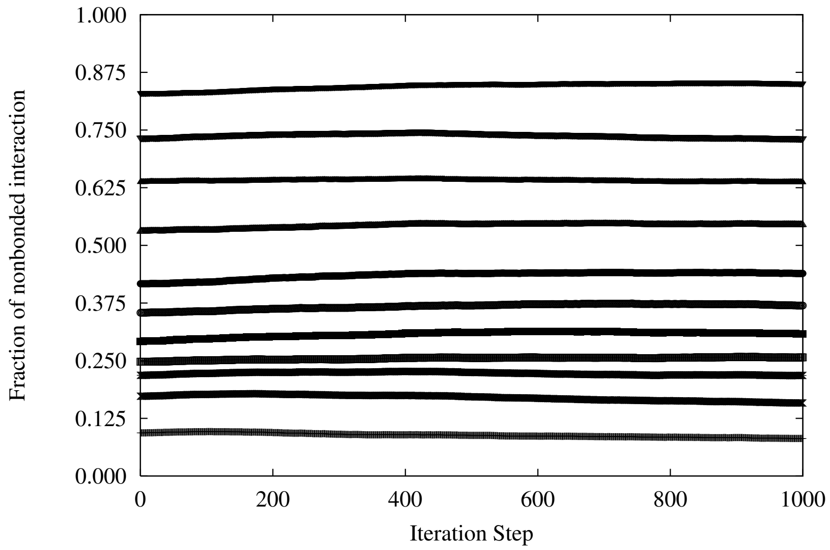

Fig. 7. Incremental fraction of nonbonded interactions calculated by each processor, with a total of twelve processors. The y-axis indicates the total load, as a fraction with each line (of the graph) indicating the fraction of load due to addition of one processor (as we go up the y-axis).

Acknowledgments

We would like to thank Darshan Thaker for his help during the experimentation process. The computations were carried out in the Parallel and Dis-

tributed Computing Laboratory in the Electrical and Computer Engineering Department at the Wayne State University.

References

[1] Janak J.F. and Pattnaik P.C., *J. Comp. Chem.*, **13** (1992) 533.

[2] Janak J.F. and Pattnaik P.C., *J. Comp. Chem.*, **13** (1992) 1098.

[3] Jin R.Y., Pernice M. and Boyd R.H., *Comp. and Theo. Polym. Sci.*, **6** (1996) 9.

[4] Clark T., Hanxlenden R., Kennedy K., Koelbel C. and Scott L., *Proceedings of the Scalable High Performance Computing Conference*, (1992) 98.

[5] Mattson T. and Shifman M., *Proceedings of the Scalable High Performance Computing Conference*, (1992) 224.

[6] Muller-Plathe F., *Comput. Phys. Commun.*, **61** (1991) 285.

[7] Plimpton T. and Hendrickson B., *J. Comp. Chem.*, **17** (1996) 326.

[8] Liem S.Y., Brown D. and Clarke J.H.R., *Comput. Phys. Commun.*, **67** (1991) 261.

[9] Brown D., Clarke J.H.R., Okuda M. and Yamazaki T., *Comput. Phys. Commun.*, **83** (1994) 1.

[10] Brown D., Clarke J.H.R., Okuda M. and Yamazaki T., *Comput. Phys. Commun.*, **74** (1993) 67.

[11] Eisenhauer G and Schwan K., *J. Parallel and Distributed Computing*, **35** (1996) 76.

[12] Nyland L., Prins J., Yun R.H., Hermans J., Kum H. and Wang L., *J. Parallel and Distributed Computing*, **47** (1997) 125.

[13] Plimpton S., *J. Comp. Phys.*, **117** (1995) 1.

[14] Murty R. and Okunbor D., *Parallel Computing*,**25** (1999) 217 .

[15] Hendrickson B. and Plimpton S., *J. Parallel and Distributed Computing*, **27** (1995) 15 .

[16] Brown D., Minoux H. and Maigret B., *Comput. Phys. Commun.*, **103** (1997) 170.

[17] Taylor V.E., Stevens R.L. and Arnold K.E., *J. Parallel and Distributed Computing*, **45** (1997) 166.

[18] Jin R.Y., Song K.H. and Hase W.L., *J. Phys. Chem.*, submitted.

[19] Allen M. and Tildesley D., 'Computer simulation of liquids', Clarendon Press, Oxford Science Publications, (1987).

[20] Berendsen H.J.C, Postma J.P.M., van Gunsteren W.F., DiNola A. and Haak J.R., *J. Chem. Phys.*, **66** (1984) 2821.

[21] Swope W.C., Andersen H.C., Berens P.H. and Wilson K.R., *J. Chem. Phys.*, **76** (1982) 637.