COMPUTERS AND BIOMEDICAL RESEARCH 25, 168–180 (1992)

Molecular Dynamics Simulation on a Network of Workstations Using a Machine-Independent Parallel Programming Language

MARK A. SHIFMAN

Center for Medical Informatics and Department of Anesthesiology, Yale University School of Medicine, New Haven, Connecticut 06510

ANDREAS WINDEMUTH AND KLAUS SCHULTEN

Beckman Institute and Department of Physics, University of Illinois, Urbana-Champaign, Illinois 61801

AND

PERRY L. MILLER

Center for Medical Informatics and Department of Anesthiology, Yale University School of Medicine, New Haven, Connecticut 06510

Received April 4, 1991

Molecular dynamics simulations investigate local and global motion in molecules. Several parallel computing approaches have been taken to attack the most computationally expensive phase of molecular simulations, the evaluation of long range interactions. This paper reviews these approaches and develops a straightforward but effective algorithm using the machine-independent parallel programming language, Linda. The algorithm was run both on a shared memory parallel computer and on a network of high performance Unix workstations. Performance benchmarks were performed on both systems using two proteins. This algorithm offers a portable cost-effective alternative for molecular dynamics simulations. In view of the increasing numbers of networked workstations, this approach could help make molecular dynamics simulations more easily accessible to the research community.

INTRODUCTION

Molecular dynamics, the study of local and global motion in molecules, has become increasingly important for the investigation of structure–function relationships in biological molecules (*1*). Typical local motions include move-

0010-4809/92 $3.00

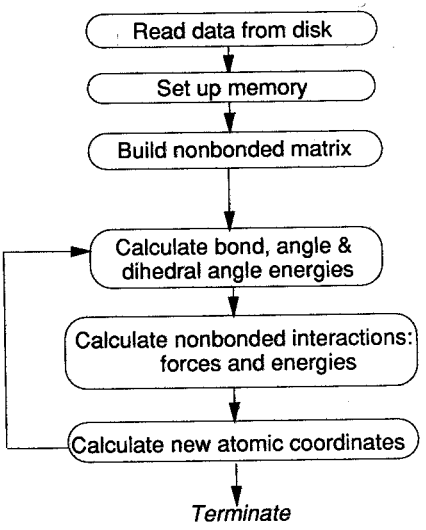


FIG. 1. Sequential molecular dynamics algorithm. This figure gives an overview of the computation for a typical molecular dynamics algorithm. The majority of the computation time is spent in the calculation of nonbonded forces and energies.

ments of an amino acid or groups of amino acids in the binding site of a protein, while typical global motions include movement of globular domains about hinge regions. Since the direct experimental measurement of dynamic behavior is extremely difficult, molecular simulations have been employed to study these problems.

A number of algorithms have been proposed to allow molecular dynamics simulations to run on parallel computers. In this paper, we review the computational strategies for parallelizing molecular simulations and describe a portable algorithm for parallelizing these simulations. This algorithm has been implemented using Linda, a machine-independent parallel programming language, on both a network of Unix workstations and on a shared memory parallel computer. The network implementation gives molecular dynamics simulations easy access to a very powerful computing resource available at many academic institutions, a network of high performance workstations.

MOLECULAR DYNAMICS SIMULATIONS

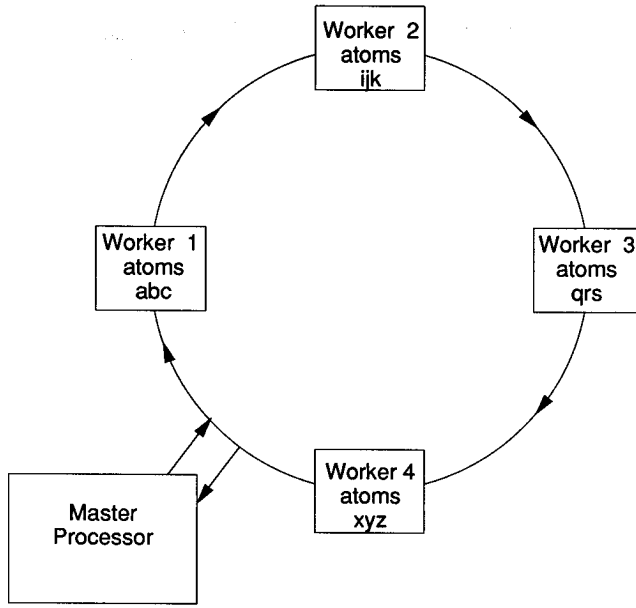
Molecular simulations involve the iterative computation of the total potential energy, the forces and updated coordinates for each atom in the molecule. A schematic overview of the sequential steps for a molecular dynamics simulation is shown in Fig. 1. The total potential energy typically includes bond energy, angle energy, dihedral angle energy, and nonbonded energy which includes van der Waals and electrostatic terms. Each of the energies can be relatively rapidly calculated except nonbonded energy. Updating the coordinates is performed

by numerically solving the Newtonian equations of motion, which is a relatively inexpensive computation.

The calculation of nonbonded energy and corresponding force terms (long-range interactions that occur between pairs of atoms) is extremely computationally intensive since contributions may have to be calculated for each pair of atoms. Nonbonded interactions are usually not calculated for atoms that are covalently bonded to each other or that are separated by only two covalent bonds. The atoms that do interact via nonbonded forces are typically indicated in a matrix, the nonbonded matrix. Since the force of atom i on atom j is the exact inverse of the force of atom j on atom i, the interaction matrix is symmetric and can be stored as a triangular matrix. One approach to reducing the computation time is to exclude the calculation of nonbonded forces if the distance between a pair of atoms is greater than some cutoff distance, although some error is introduced when a distance cutoff is employed. The information on such exclusions can also be encoded as part of the nonbonded matrix which may then require periodic updating as the atoms move relative to each other.

Several parallel computing approaches have been taken to attack the most computationally expensive phase of molecular simulations, the evaluation of long-range interactions. Vector supercomputers are frequently employed for these simulations (*2*). These computers gain their speed by performing operations on a vector of data in parallel. For example, the difference between two vectors, (X_1,X_2,X_3) and (Y_1,Y_2,Y_3), can be calculated by having $(X_1 - Y_1)$, $(X_2 - Y_2)$, and $(X_3 - Y_3)$ calculated simultaneously rather than sequentially calculating the difference in the first, then second, and then the third components on one processor. This approach is effective for molecular simulations where many vector calculations with atomic coordinates, velocities, and forces are performed. Although parallelized vector calculations on supercomputers are very fast, supercomputers are expensive to use and may not be easily accessible to the general biologic community.

A second parallel computing strategy involves building application specific hardware for performing molecular simulations (2–5). A popular approach utilizes transputers, parallel computers whose the processors can be connected in various topologies such as rings (Fig. 2). Here worker processors are arranged in a ring and can communicate with their neighbors. Each worker processor is assigned an equal number of atoms from the protein. All information required for calculation of nonbonded forces and energies on these atoms, such as position, mass, and charge, is sent to the workers. Each worker computes the nonbonded forces and energies between its local atoms. The nonbonded interactions between atoms assigned to a given worker and atoms assigned to its neighbor in the clockwise direction are computed by passing a copy of the atomic information to the nearest neighbor which then computes the contribution of its local atoms with the atoms passed to it. Each copy of atomic information visits each processor in the ring with nonbonded interactions being calculated at each step. The forces are then collected by the master process and used for computing updated atomic coordinates. This algorithm implements



Step	Worker 1	Worker 2	Worker 3	Worker 4
1	abc	ijk	qrs	xyz
	abc	ijk	qrs	xyz
2	abc	ijk	qrs	xyz
	xyz	abc	ijk	qrs
3	abc	ijk	qrs	xyz
	qrs	xyz	abc	ijk
4	abc	ijk	qrs	xyz
	ijk	qrs	xyz	abc

FIG. 2. Parallel molecular dynamics algorithm with processors distributed in a ring. Each worker processor is assigned an equal portion of the molecule. A copy of the data describing atoms in each worker is made for communication around the ring. First the interactions of the atoms within that copy are computed. The copies are then rotated to the next processor (step 2) and the nonbonded interactions are computed with the local atoms of that worker. This process is continued until the copies traverse the entire ring. The algorithm includes provisions for avoiding duplicate computations as the copies traverse the ring.

parallelization at a higher level than vectorization by decomposing the main problem into several similar tasks, each of which is allocated to a different processor. Transputers connected in a ring topology have been shown to be extremely competitive and cost effective relative to supercomputers; however, the computer itself must be constructed and links must be physically configured by hardware specialists thus limiting broad accessibility of the approach.

General purpose parallel computers have also been utilized for molecular simulations. Hypercubes are distributed memory parallel computers which can be configured with the processors in a ring. These parallel computers have been used with algorithms comparable to that used on transputers [6]. The logic to coordinate parallel computation on a hypercube is programmed with low level, machine dependent, function calls, thus making the code difficult to transport to other platforms. A modified ring algorithm, the replicated systolic loop algorithm, has also been implemented on the Connection Machine, a parallel computer with a large number of very simple processors [7].

Shared memory parallel computers offer the potential for a different algorithmic strategy [8]. Global information about the molecule such as the current atomic coordinates can be stored in shared memory which is accessible to all processors. This architecture also facilitates the use of a nonbonded exclusion matrix which can also be stored in shared memory.

A schematic representation of the shared memory algorithm is shown in Fig. 3. The shared memory algorithm includes a master process which sets up the computation, starts workers, and collects results from workers. Each worker calculates the nonbonded interactions on portions of the molecule. The atomic coordinates and nonbonded matrix are globally available, therefore this data does not need to be passed from processor to processor. The partitioning of the computation must take into account the fact that different portions of the molecule require varying amounts of computation to prevent some workers laying idle while others are still working. This "load balancing" can be accomplished dynamically by dividing the molecule into many small tasks relative to the number of workers. Workers then accept and perform tasks until all the tasks are completed. After calculating the bonded forces, the master may also be employed on worker tasks. Another approach, static load balancing to be discussed below, involves preallocating nearly equal amounts of computation to each worker.

Portability is again an issue with shared memory programs because they are highly dependent on machine architecture. Machine dependent subroutines for allocating shared memory, starting, synchronizing, and terminating workers also create portability problems. Molecular dynamics simulations could be made available to a wider research community if a portable, hardware independent strategy could be employed. A promising approach is the implementation of these simulations in Linda.

Linda for Molecular Dynamics

Linda is a computer language for implementing machine-independent parallel computations (*9, 10*). It is portable and runs on several different architectures including shared memory and distributed memory machines. The idea behind Linda is conceptually very simple. The main features include an abstract object, "tuple space," where data that is accessible to all processors can be placed. Access and modification on data in tuple space is possible with four operators,

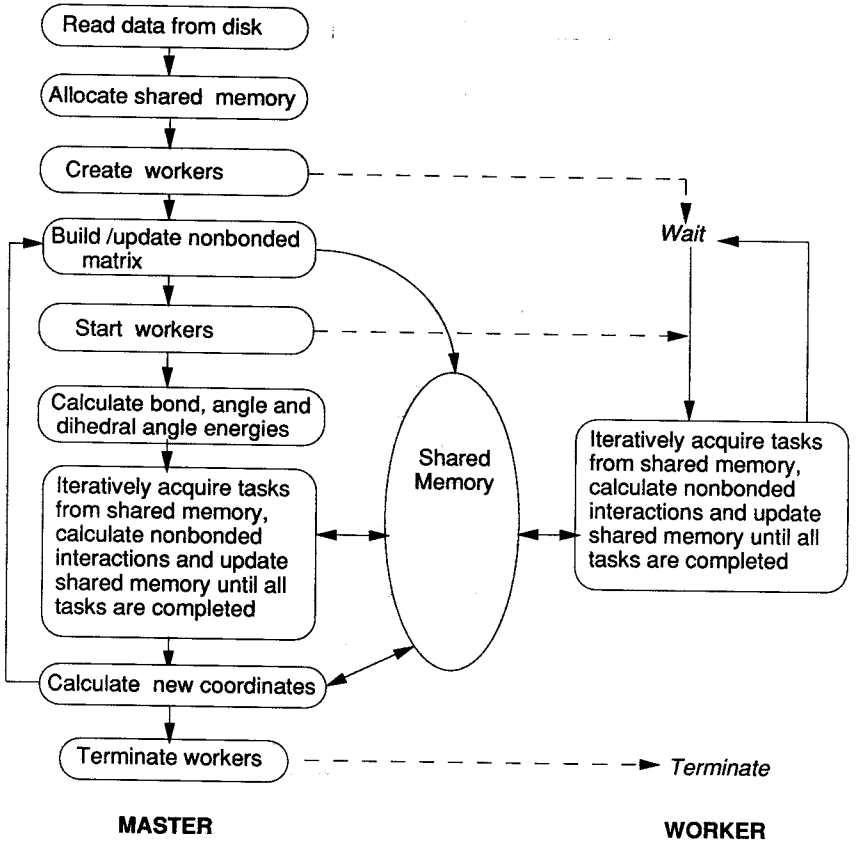


FIG. 3. Parallel molecular dynamics algorithm on a shared memory computer. This algorithm uses shared memory for global data access. The double headed arrows indicate bi-directional flow of data. The workers independently calculate nonbonded forces and energies using the nonbonded matrix and coordinates in shared memory. Nonbonded forces and energies are communicated back to shared memory. In implementations that employ a cutoff for calculating nonbonded forces, the nonbonded matrix may require updating as the simulation progresses. This figure is adapted from Fig. 1 of (*8*).

in, *out*, *rd*, and *eval*. Data in tuple space are organized as ordered tuples much like data in a relational database table. Data tuples are added to tuple space via the *out* operator. Data are requested and removed from tuple space by using the *in* operator. The tuple requested by an *in* operation must match a tuple in tuple space. If a matching tuple is not available, the requesting process waits until one is available. The *rd* operation is comparable to the *in* except the matching tuple is not removed from tuple space, i.e., a carbon copy is given to the requesting process. Finally, new processes are created with the *eval* operator.

The Linda paradigm, which is implemented by integrating the four operators

into a conventional language such as C or Fortran, fosters high level problem decomposition into parallel components. Tuple space can be used as both a highway for message passing and as a globally accessible data store.

Recently, an implementation of Linda, Network Linda, has been developed to run Linda programs on networks of high speed Unix-based computers such as Sun desktop workstations (*11*). Many institutions have large networks of powerful workstations many of which spend much of their time either completely idle or being used for light computational tasks such as text editing. Network Linda makes this massive computing resource available for parallel attack on computationally intensive problems such as molecular simulations.

The purpose of this study was two fold, to develop a new platform independent algorithm for calculating long range force interactions and to investigate the feasibility and utility of this algorithm implemented in Linda and on Network Linda.

Description of Algorithm

One standard approach to parallelizing computation using Linda includes one master process and multiple worker processes. Generally, the master sets up the computation, starts the workers, parcels out tasks, and collects the final results. The workers perform their tasks and send results either to the master for tallying or to other workers for further processing. The challenge is to balance the work so that the workers are kept busy while minimizing the amount of interprocess communication among the workers and the master.

The current algorithm uses their master–worker paradigm to parallelize a C language molecular dynamics package, MD, developed by the Theoretical Biophysics Group at the University of Illinois. A schematic representation for the parallel algorithm is shown in Fig. 4. The master starts the workers, reads from the disk files the structural information and initial coordinates for the macromolecule, and sends out global data such as parameters for calculating van der Waals interactions. The master then uses a simple heuristic, described below, to balance the workload and sends out partitioned data to each worker for calculating the nonbonded interaction matrix. The workers each read in the global parameters and build their portion of the nonbonded interaction matrix. Each worker only requires a small portion of the nonbonded matrix which is calculated locally.

A round of calculation begins with the master calculating bond, angle, and dihedral energies. These energy terms are calculated so rapidly that parallelization is not needed. Only 3–5% of the time for the sequential algorithm is spent in calculating these. The master then sends out the current coordinates of the atoms to all workers. The workers calculate partial force and energy updates for their portion of the molecule and send these back to the master where the updates are collected and used for solving the equations of motion. The updated coordinates are then used for the next round of the simulation.

The heuristic for load balancing involves giving each worker approximately

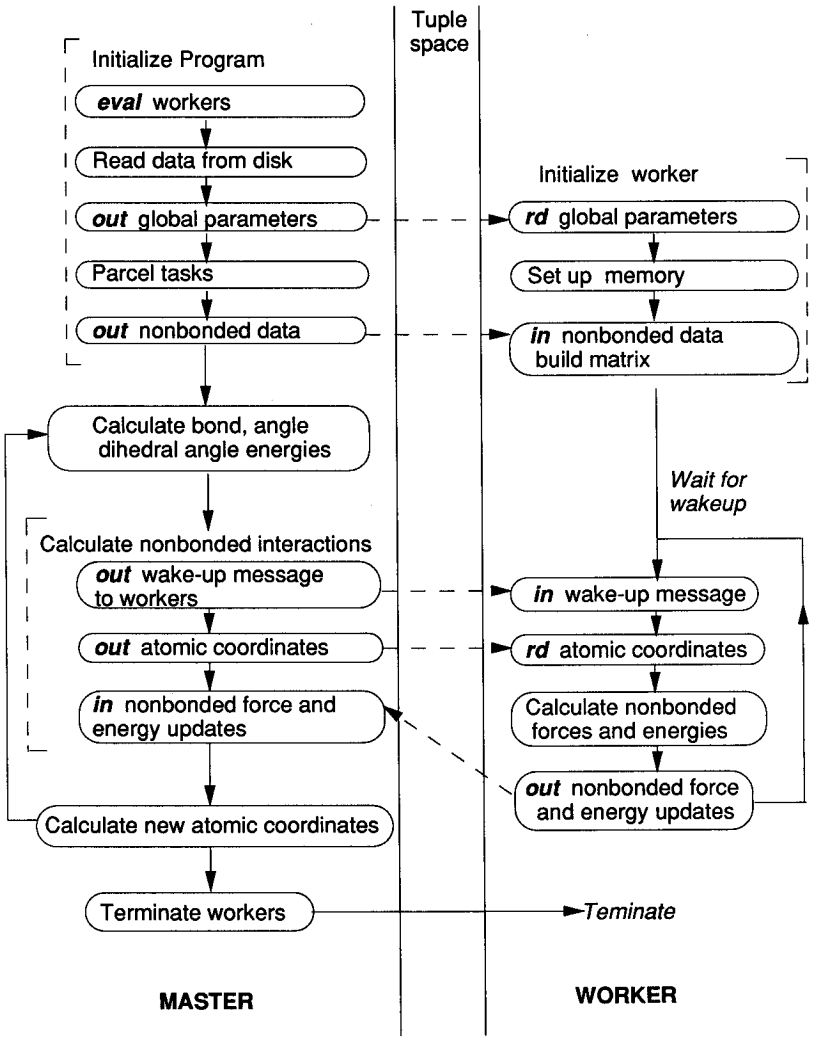
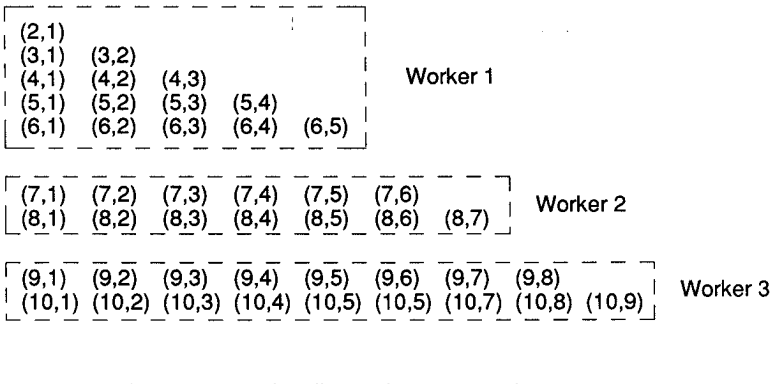


FIG. 4. Parallel molecular dynamics algorithm in Linda. This algorithm is initialized by creating workers and by communicating global parameters via tuple space to the workers. Each worker builds the nonbonded matrix for their portion of the molecule. The master calculates bond, angle, and dihedral angle energies which are relatively less computationally intensive. The workers calculate nonbonded forces and energies after acquiring current coordinates for their portion of the molecule. Nonbonded forces and energies are then sent back to the master which accumulates the updates and calculates updated coordinates by solution of Newtonian equations of motion.



worker #	pairs allocated	atom pairs
1	15	(2,1) - (6,5)
2	13	(7,1) - (8,7)
3	17	(9,1) - (10,9)

FIG.5. Heuristic for load balancing. The nonbonded matrix is a triangular matrix for a 10-atom simulation. A given pair, (i,j), indicates that a nonbonded interaction should be calculated between atom i and atom j. For simplicity, all pairwise interactions are represented except self interactions. (Atom i does not interact with itself.) The total number of atom pairs is $(10 * 9/2) = 45$. With three workers the target value is therefore $45/3 = 15$ pairs per worker. Consecutive rows are allocated to a worker until the number of elements in a row exceeds the target. The last worker gets all of the rows that are left.

the same number of pairwise interactions to calculate. If there are N atoms, there are at most $N(N\text{-}1)/2$ pairs for which interactions must potentially be calculated. (The actual number of pairs is smaller because covalently bonded atoms are excluded.) The total number of pairs is divided by the number of workers to obtain a target value of pairs per node. Rows of the triangular nonbonded interaction matrix are consecutively allocated to a worker until the accumulated number of pairs for the node exceeds the target value of pairs per node. This approach to load balancing is quite effective when the number of workers is much smaller than the number of atomic pairs. In fact, all workers generally finish within 2–3 sec of each other, which is about 1% of the total computation time. A simple example of this heuristic is presented in Fig. 5.

Our algorithm is conceptually similar to the shared memory algorithm except the global data is passed through tuple space to the workers rather than being read from shared memory. Our algorithm also uses static load balancing rather than dynamic load balancing, but could easily be modified to perform dynamic load balancing. The master would allocate tasks consisting of a portion of the macromolecule with its associated coordinates and nonbonded matrix. The workers would then acquire tasks and calculate the nonbonded interactions until all tasks are completed. This algorithm is simpler than the distributed algorithms, which pass portions of the macromolecule around the ring, and involves much less interprocess communication.

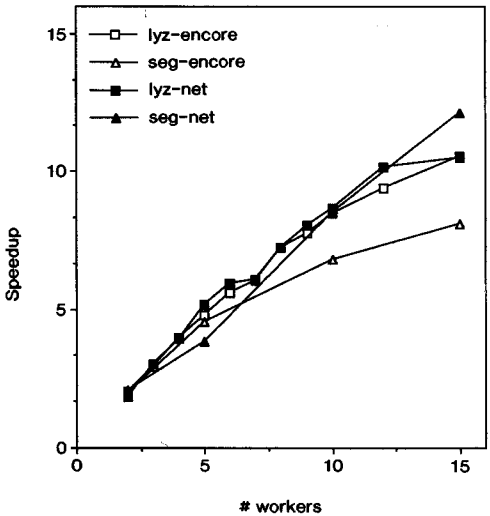


FIG. 6. Speedup vs. Number of Workers on a shared memory computer and on Network Linda. The speedup is plotted vs. the number of workers on both the Encore Multimax and on Network Linda running on Sun SparcStations. Two proteins, lysozyme and a segment of the reaction center of *Rhodopseudomonas viridis*, were studied.

METHODOLOGY, RESULTS, AND DISCUSSION

The original sequential C language program, MD, for calculating nonbonded energy and force components was developed by the Theoretical Biophysics Group at the University of Illinois. This program was parallelized by adding Linda operations for the above algorithm described above (*9, 10*). The program was then run on both the Encore Multimax shared memory parallel computer and on a network of Sun workstations using Network Linda. Two proteins were studied: chick lysozyme containing 1265 atoms and a segment of the photosynthetic reaction center of *Rhodopseudomonas viridis* containing 3634 atoms. The number of processors was varied to determine the speedup and efficiency of the algorithm on both parallel computer systems. The base time against which all times were compared was the time for the parallel program running with the master and one worker. This time for one worker was essentially the same as that for the sequential program. (One iteration of the sequential program using lysozyme took 56 sec on a Sun workstation vs. 54 sec with one worker on Network Linda.) Each experiment was performed five times and the runtimes averaged. Speedup was calculated by dividing the base time by the time obtained with multiple workers. Efficiency was calculated by dividing speedup by the number of workers (*10*). Notice that this definition of efficiency does not include the master process.

Figure 6 shows a plot of the speedup vs. the number of workers obtained for lysozyme and segment of *R. viridis* protein on the Encore and on Network

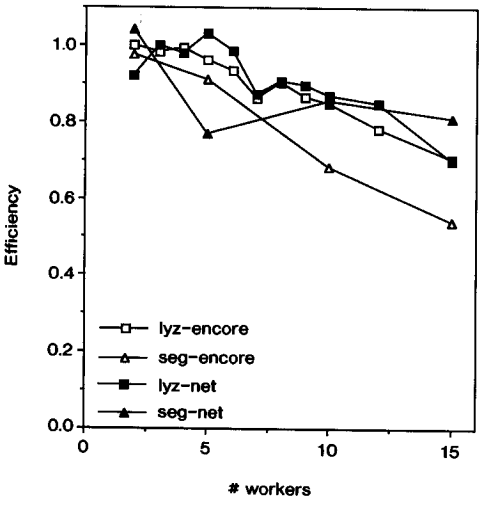


FIG. 7. Efficiency vs. Number of Workers on a shared memory computer and Network Linda. The efficiency is plotted vs. the number of workers for both the Encore Multimax and for Network Linda. Efficiency is calculated by dividing the speedup by the number of workers. The proteins are the same as those in Fig. 6.

Linda. There is essentially linear speedup with increasing numbers of nodes on both systems. Figure 7 shows the efficiency vs. the number of nodes for both proteins. Both systems show a mild decrease in efficiency with increasing the number of nodes. The performance of this algorithm is competitive with that recently reported for a shared memory algorithm where the efficiency of 0.66 was obtained on a Hewlett-Packard/Apollo DN10000 with four processors (*8*). Our efficiency on a shared memory computer with four workers was 0.99 and 0.83 for lysozyme and the segment the *R. viridis* protein.

A respresentation calculation for the segment of the *R. viridis* protein on the Encore took 1862 sec with one worker. This time was reduced to 230 sec with 15 workers. This is a speedup of 8 and an efficiency of 0.54. On Network Linda the times were 553 sec for one worker and 45 sec for 15 (speedup = 12, efficiency = 0.81). The speed difference between the two systems with one worker is due to the faster processor speed of the Sun workstation.

The loss in efficiency with increasing numbers of workers may have several causes. Interprocess communication overhead is a major factor. The results were quite similar on both a shared memory computer and Network Linda. This is interesting since it implies the communication time on the ethernet is still very small compared to the time of calculating the force components. The simulation for the segment of the *R. viridis* protein on the Encore was moderately less efficient than on Network Linda. This may be due to bus traffic and memory management on the shared memory machine, since the memory requirements were significantly larger for this protein. With Network Linda, each processor

had 16 or more Mbytes of memory, thereby allowing each processor to handle a large amount of data if necessary.

There are several advantages to our algorithm implemented in Linda. Portability is clearly demonstrated since identical source code was run on both a shared memory parallel machine and a distributed network of computers. Also, Linda has been implemented on a variety of other parallel computers, including the Intel Hypercube. Although parallelization strategy is relatively straightforward and easy to understand, the performance is competitive with other published algorithms.

Implementing molecular dynamics simulations on Network Linda is appealing for several reasons. One has access to many very powerful processors, in theory as many processors as there are workstations on the network. At our institution, there are plans to incorporate well over 50 Sun workstations into the Network Linda environment. As the number of workstations increases the potential computational power of Network Linda will surpass the power of a supercomputer. Another advantage may exist for problems where the data structures are too large to conveniently fit onto single workstation. These problems can be easily partitioned and run on distributed workstations which collectively have more memory than a single workstation or shared memory computer.

In summary, we have developed a platform independent algorithm for molecular dynamics simulations which incorporates parallelization for the computation of long-range interactions. The Linda approach appears promising for addressing these computationally intensive problems. Furthermore, Network Linda offers an accessible, powerful, cost-effective alternative for molecular simulations.

ACKNOWLEDGMENTS

This research was funded in part by NIH Grants T15 LM07056 and R01 LM05044 from the National Library of Medicine. This paper is an expanded version of a paper published in the 1991 15th Annual Symposium on Computer Applications in Medical Care (SCAMC), reprinted with permission.

REFERENCES

1. McCammon, J. A., and Harvey, S. "Dynamics of Proteins and Nucleic Acids." Cambridge Univ. Press, Cambridge, UK, 1987.
2. Fincham, D. Parallel computers and molecular simulation. *Mol. Simul.* **1,** 1 (1987).
3. Heller, H., Grubmuller, H., and Schulten, K. Molecular dynamics simulation on a parallel computer. *Mol. Simul.*, in press.
4. Li, J., Brass, A. Ward, D. J., and Robson, B. A study of parallel molecular dynamics for N-body simulations on a transputer system. *Parallel Comput.* **14,** 211 (1990).
5. Ellingworth, H. R. P. "Parallel Algorithms for the Force-Field Method in Molecular Modelling." Ph.D. dissertation, Oxford University, 1989.
6. Fox, G. C., Johnson, M. A., Lyzenga, G. A., Otto, S. W., Salmon J. K., and Walker, D. W. "Solving Problems on Concurrent Processors." Vol. I. Prentice–Hall, Englewood Cliffs, NJ, 1988.

7. WINDEMUTH, A., AND SCHULTEN, K. Molecular dynamics simulation on the Connection Machine. *Mole. Simul.* **5,** 353 (1991).
8. MULLER-PLATHE, F. Parallelising a molecular dynamics algorithm on a multi-processor workstation. *Comput. Phys. Commun.* **61,** 285 (1990).
9. CARRIERO, N. J., AND GELERNTER, D. H. Linda in context. *Commun. ACM* **32,** 444 (1989).
10. CARRIERO, N. J., AND GELERNTER, D. H. "How to Write Parallel Programs: A First Course." MIT Press, Cambridge, MA, 1990.
11. GELENTER, D. H., AND PHILBIN, J. Spending your free time. *Byte* **15,** 213 (1990).