A Linked-Cell Domain Decomposition Method for Molecular Dynamics Simulation on a Scalable Multiprocessor

L. H. YANG, E. D. BROOKS III, AND J. BELAK

Massively Parallel Computing Initiative, Lawrence Livermore National Laboratory, Livermore, CA 94551

ABSTRACT

A molecular dynamics algorithm for performing large-scale simulations using the Parallel C Preprocessor (PCP) programming paradigm on the BBN TC2000, a massively parallel computer, is discussed. The algorithm uses a linked-cell data structure to obtain the near neighbors of each atom as time evolves. Each processor is assigned to a geometric domain containing many subcells and the storage for that domain is private to the processor. Within this scheme, the interdomain (i.e., interprocessor) communication is minimized. © 1993 by John Wiley & Sons, Inc.

1 INTRODUCTION

The molecular dynamics (MD) computer simulation method [1] is a well-established and important tool in the fields of physics, chemistry, biology, and engineering. Recent advances in computer hardware and software techniques have allowed the simulation of large systems [2] using realistic potential models [3, 4]. Given that the requirement for large-scale and more realistic MD simulations will remain, the demand for advanced hardware and software architecture will continue to grow into the future. In anticipation of this future demand, we present an algorithm for performing large-scale MD simulations of simple metals with short-range embedded-atom interatomic potentials, on a scalable multiprocessor, the BBN TC2000. The current configuration of the TC2000 at Lawrence Livermore National Laboratory (LLNL) consists of 128 Motorola 88100 reduced instruction set microprocessors running at 20 MHz. Each processor is located on a separate board, or node, along with a 16-kilobyte cache and 16 megabytes of local memory. The nodes are interconnected by a scalable switch. At boot time, some of each processor's local memory is dedicated to an interleaved shared memory pool. Thus, allocated memory is either private (accessible by only one processor making a memory reference within its own node) or shared (accessible by all processors via a memory reference through the switch).

The method of MD simulation involves the evaluation of the force acting on each atom due to other atoms and the numerical integration of the

Received February 1992
Revised October 1992

newtonian equations of motion. The most time-consuming portion of any MD simulation is the evaluation of the force; it typically consumes up to 98% of the total simulation time. In general each atom interacts with every other atom in the system. However, for many physical systems such as rare gases and simple metals, the interaction is non-negligible for neighboring atoms within a few atomic diameters only, and the computational demands are drastically reduced. Given that the neighbors of an atom do not change appreciably during a short period of simulation time, two general methods have been developed to keep track of an atom's neighbors. In the neighbor-list approach [5], all atoms within a sphere surrounding a given atom are stored in a list. The radius of the sphere is chosen to exceed the interaction range by an amount sufficient to permit updating of the neighbor list at intervals of several MD time-steps only (typically, 10–20 steps). The neighbor lists provide a natural mechanism for developing vectorized MD algorithms [6]. However, the large memory overhead associated with maintaining these lists prohibits usage in large-scale simulations ($>10^6$ atoms). In the linked-cell methods, the system is subdivided into many small cells and linked-lists [7] of atoms belonging to each individual cell are constructed. Because of the random addressing of memory, linked-list algorithms do not vectorize efficiently and extensive efforts have gone into vectorizing large-scale MD algorithms [8]. The number of atoms per linked cell depends on the interaction range and the software architecture (whether the interaction is limited to neighboring subcells only or further subcells are allowed to interact). Unlike the neighbor-list approach, the cell lists need to be renewed at each MD time-step. The low memory overhead associated with the linked-list is ideal for large-scale MD simulation and the cell-list algorithm introduces a "natural" domain decomposition parallelism into the MD problem [9]. In this paper, we describe our effort to adapt this domain decomposition scheme to our linked-cell MD code [10] in order to improve the parallel efficiency on the BBN TC2000. This scheme also allows for easy porting to machines supporting the message passing programming model only.

In general, a three-dimensional system of N atoms of mass m in a simulation cell of sides L_x, L_y, and L_z is the basic building block of an MD system. We employ an embedded-atom method (EAM) [3] to express the interaction between the atoms in a simple metal. The total potential energy is written as:

$$U_{total} = \frac{1}{2} \sum_{i,j \neq i} \varphi(r_{ij}) + \sum_i F(\rho_i). \tag{1}$$

with

$$\rho_i = \sum_{j \neq i} f(r_{ij}). \tag{2}$$

The first term is the usual two-body interaction energy and the second term $(F(\rho_i))$ is the energy required to embed the atoms into the local electronic charge density (ρ_i), which is written as a superposition of charge densities due to neighboring atoms $(f(r_{ij}))$. The newtonian equations of motion for the EAM are

$$m \frac{d^2 x_k}{dt^2} = - \sum_{j \neq k} \{\varphi'(r_{kj}) + (F'(\rho(r_j)) + F'(\rho(r_k)))f'(r_{kj})\} \frac{x_k - x_j}{r_{kj}}. \tag{3}$$

These equations are inherently nonlocal: they depend on both the embedding densities at atom-$k(\rho(r_k))$ and at atom-$j(\rho(r_j))$. They must be solved in a two-step manner. The embedding density at all atomic sites r_j is evaluated first, then the forces acting on each atom may be calculated.

The equations are integrated by approximating the time derivative with a central difference [5]:

$$\frac{d^2 x}{dt^2} \approx \frac{x(t + \Delta t) - 2x(t) + x(t - \Delta t)}{\Delta t^2}, \tag{4}$$

where the time-step (Δt) is 1/25 of the vibration period (τ_E). For a simple metal τ_E is about 0.3×10^{-12} seconds.

Domain decomposition techniques have received much attention recently, as they are suited for the parallel implementation of problems with localized data. The local nature of the short-range potential provides a good opportunity to apply the domain decomposition scheme to our MD simulation on a scalable multiprocessor machine. In general the physical MD cell is divided into geometrically separate subdomains, each containing many subcells. Each processor is assigned to a subdomain and is responsible for updating all atoms contained therein. We employ the shared memory on the TC2000 as a "hub" for data communication between subdomains, although the same communication/decomposition algorithm may also be implemented using the message-passing programming model on distributed memory machines [11–13].

The outline of this paper is as follows. In Section 2 we describe the basic data structures in our MD code and the programming model used on the BBN TC2000. In Section 3 we discuss our implementation of the linked-cell list method and the domain decomposition scheme on the BBN TC2000. Performance benchmark results are presented and discussed in Section 4.

2 THE MD PROGRAM AND THE PARALLEL C PREPROCESSOR (PCP) PROGRAMMING MODEL

Our MD program is designed to study various types of tribological systems (e.g., friction and wear). Details concerning the applications of MD simulations to these problems can be found in the papers by Belak and Stowers [14, 15]. Figure 1 illustrates a schematic geometry of a typical system. For the simulation of the orthogonal metal cutting process, the MD simulation cell is a fixed window in the reference frame of the tool. The

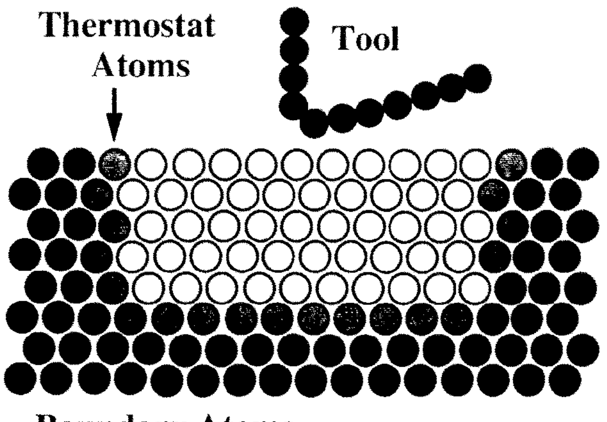

FIGURE 1 The geometry of our steady-state variable-particle molecular dynamics model of orthogonal metal cutting. The calculation is performed within the reference frame of the tool. The thermostat atoms are maintained at room temperature and the boundary atoms are used to impose the cutting speed—they propagate to the right at the cutting speed.

boundary atoms are used to impose the cutting speed; they move to the right at the cutting speed. In order to produce a steady state flow, new atoms are continuously inserted from the left, while atoms that leave the top or the right of the cell are discarded: the system is open. Next to the boundary, we place a thermostat region. A time-dependent viscous damping (ζ) is added to the equation of motion for the atoms in this region [16, 17].

$$m\,\frac{d^2x_k}{dt^2} = force - \zeta m\,\frac{dx_k}{dt}, \qquad (5)$$

with

$$\frac{d\zeta}{dt} = \frac{1}{\tau^2}\left(\frac{T_{calc.}}{T_{desired}} - 1\right). \qquad (6)$$

Where τ is the relaxation time for ζ and T_{calc} is the kinetic temperature of the system. The purpose of the thermostat is to remove heat from doing work at the tool tip.

Our MD computer simulation code is written in the C programming language. Unlike Fortran, C provides the advantages of allowing complex data structures, pointer arithmetic, and dynamic allocation of memory, all of which are necessary for performing variable atom simulations. We employ the PCP [18] as our programming model on the BBN TC2000. PCP provides an extension of the single-program-multiple-data (SPMD) programming model in the familiar C programming environment. Each processor executes the same code and the path through that code is determined by the data that the processor encounters. PCP introduces the concept of a "team" of processors. Each team has one master processor that is used for performing serial work such as data accumulation and initialization. Flow synchronization is obtained through the `barrier` statement. Every processor reaching a barrier waits until all members of its team (including the master) reach that barrier. Additional flow control for critical sections is accomplished with locks. PCP provides the `lock (&lock_variable)` and `unlock (&lock_variable)` functions to isolate critical sections. The lock variable is stored in shared memory. The first processor entering the critical section sets the lock variable to locked and proceeds with the calculation. Meanwhile, the remaining processors test the lock variable to see whether it is locked. When the first processor finishes the calculation, it sets the lock variable to unlocked. The next processor to find it unlocked immediately locks it and proceeds with the calculation.

Parallelism is exploited via domain decomposition. Each processor does the work for its domain and the interprocessor communication is performed through the shared memory. In effect, each processor is performing a separate MD simulation and obtaining boundary data from neighboring processors. Locality of data is exploited by explicitly declaring variables as private or shared with the **private** and **shared** storage class modifiers. Private and shared memory are dynamically allocated using the **prmalloc** and **shmalloc** functions.

The majority of our parallel MD code consists of routines from a standard serial MD code based on the linked-cell method. We use the linked-cell method in our parallel domain decomposition. Before going into our domain decomposition implementation, we list the most important functions in our MD code as follows:

1. Initialization
   (a) Read input file
   (b) Initialize positions and velocities, build the linked-cell lists
2. Impose the domain decomposition scheme to divide the work for each processor
3. Main simulation loop
   (a) *force*—calculate interatomic forces
   (b) *update*—obtain new positions using the central difference
   (c) *kinetic*—calculate kinetic energy and the temperature of the system
   (d) *output*—accumulate data and output
   (e) *celler*—apply boundary conditions and update linked-cell lists

The initialization and output blocks represent a small amount of serial work that is performed by the master processor. The heart of the work throughout the program consists of loops of the following type:

```
for (i = 0; i < nx; i++) {
for (j = 0; j < ny; j++) {
for (k = 0; k < nz; k++) {
            a_ptr = cell[i][j][k].start_ptr;
            while( a_ptr != NULL)
            {
               private work
               a_ptr = a_ptr->cell_list_ptr;
         }
}}}.
```

where `cell[i][j][k].start_ptr` indicates the pointer to the first atom for each subcell and `cell_list_ptr` points to the next atom pointer within that subcell. NULL is a parameter to indicate the end of a linked-cell list. One aspect of this task we want to emphasize is that all data are private to each processor, thus the memory latency is minimized.

One may note that in the scheme we have described above, we have done the domain (and therefore the work) allocation to the processors once at the start of the job. This is suitable for a problem wherein the dynamics of the simulation do not change the load balance while the problem is executing. If our test problem did exhibit large changes in processor loading as executing proceeded, we would periodically repeat step 2 in order to maintain a good load balance for the processors.

3 PARALLEL IMPLEMENTATION

The most important aspects of a parallel implementation of the MD algorithm are the assignment of the processors to a subset of atoms and the computation of the forces on those atoms. In the following, we discuss the linked-cell list method and the domain decomposition scheme that we used in our parallel implementation.

3.1 The Linked-Cell List Method

For finite range interactions the amount of computations is reduced from $O(N^2)$ to $O(N)$, where N is the total number of atoms within the simulation cell. For example, within a simple metal, each atom has $O(100)$ neighbors that contribute to the force. We use the linked-cell list method to keep track of the distribution of atoms in each cell. Each cell has a pointer to the first atom it contains. Each atom has a pointer to the next atom in the same linked-cell. The advantage of using the linked-cell list approach in our MD code is that storage requirements are minimized. The disadvantage is that the list must be renewed at each time-step and memory is addressed in a random manner. However, the time spent updating the cell list is $O(N)$ and is found to be small relative to the force calculation.

A linked-cell list is built according to the coordinates of the atoms inside the MD simulation box. For each atom, we compute a cell index according to the current position and add the atom's pointer to the top of the list for that cell. Each cell

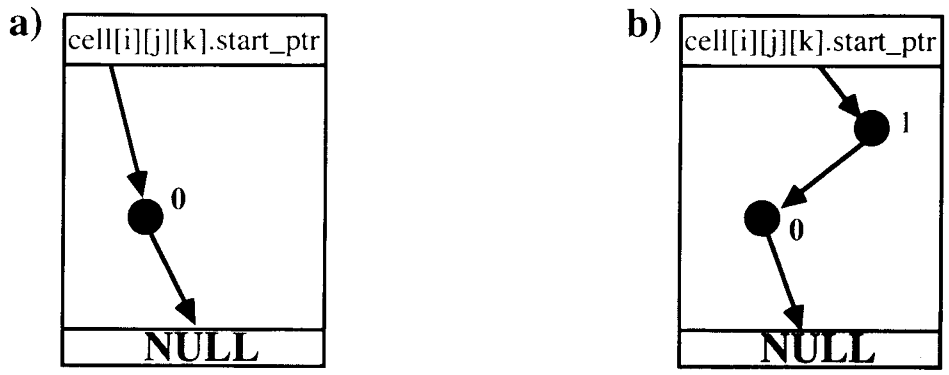

FIGURE 2 The assignment of the first two atoms to the cell-linked list. Each atom calculates its cell indices according to the atom position. (a) The first atom is assigned to cell [*i*][*j*][*k*], which has a pointer cell [*i*][*j*][*k*].start_ptr to indicate that this is the first atom on the list. In addition, this atom is linked to NULL, which is the parameter to indicate that this is the end of the cell list. (b) The second atom is added to the list by assigning it to the top of the list. By doing that, the pointer cell[*i*][*j*][*k*].start_ptr has been replaced by the second atom pointer and this atom is linked to the first atom.

has a pointer to the first atom it contains and has a flag to indicate the last atom it contains (Fig. 2). Because the cell lists are built locally in each processor, critical sections are eliminated. Critical sections were eliminated from the previous version of the code [10] by calculating both the force on atom-*i* due to atom-*j* and the force on atom-*j* due to atom-*i*, thus doubling the work to assure scalability. At the end of the time-step, any atom that migrates from one cell to another cell is removed from its cell list and is added to another cell list. In Figure 3, we illustrate how an atom is removed from the middle of a cell list and is then added to the beginning of another cell list.

3.2 Domain Decomposition

In the domain decomposition scheme, the physical domain is divided into geometrically separate subdomains, each handled by a different processor. With the aim of minimizing the need for processors to communicate with one another and in order to obtain the necessary neighbor-cell information for the force calculation, we have introduced overlapping regions (Fig. 4, the so-called exterior border subcells). As a result, each processor works independently at each MD time-step to calculate forces acting on each atom, obtain new atomic positions, and obtain new cell lists for inte-

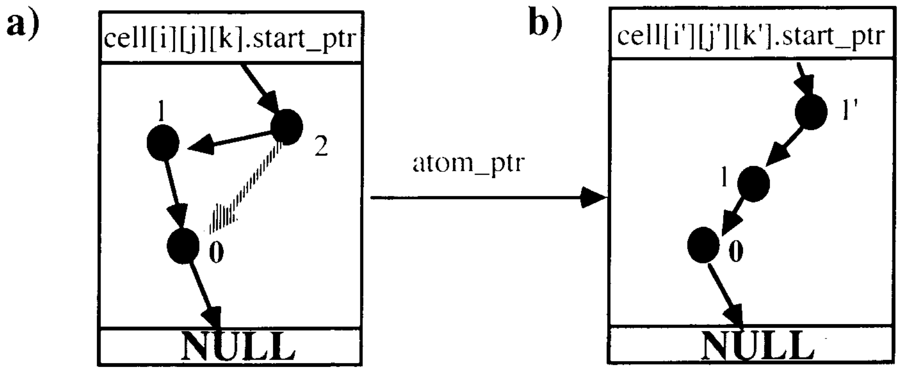

FIGURE 3 Updating the linked-cell list. Atom-1 is removed from the list by replacing the next atom pointer in atom-2's data structure with the next atom pointer from atom-1. Storage for atom-1 is returned to a buffer. (b) Atom-1′ is added to another cell list by connecting to the top of the list. Note that the atom always carries its own pointer no matter where it goes.

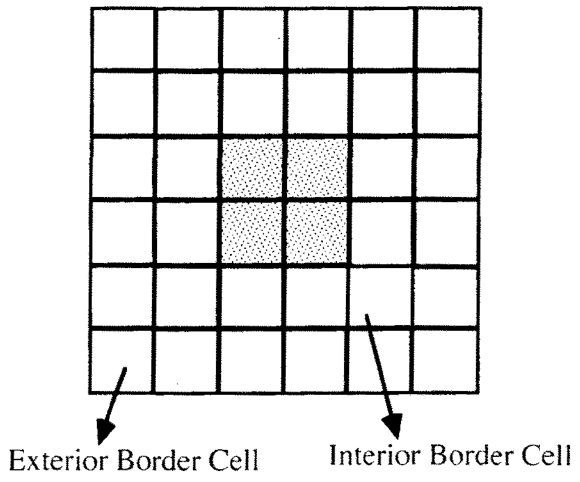

FIGURE 4 Molecular dynamics domain decomposition. At the beginning of an MD simulation, each processor, _IPROC, is assigned to a set of linked-cells (a domain). Each domain consists of a list of interior subcells (shaded squares) and exterior border subcells (empty squares). The assignment of exterior border subcells to each processor is for temporary storage used within the force calculation.

rior subcells. At the end of each time-step, interdomain communication takes place in order to update the data structure of the exterior border subcells.

The message-passing programming model may be used for performing the interdomain communication on distributed memory machines, like the TC2000 [12, 13]. Each processor knows the identity of its nearest neighbors. The data from the inner border subcells of each processor are sent to its neighboring processors. Each processor then receives messages from its neighbors and scatters the data appropriately to the outer border subcells. To ensure that the corner data are passed correctly, data are passed to neighboring processors in one direction first, then in another direction. The forces are calculated independently within each processor.

In our approach, we use the scalable shared memory facility of the BBN TC2000 to avoid the complexity of data packing and communications management that would have been required by a strict message-passing implementation. To carry out the interdomain communication, we use the shared memory as a "hub" for temporary storage of new atomic positions in the border subcells. First, atomic positions of interior border subcells from each processor are copied into share memory. Then, each processor updates its exterior border subcells by copying those atomic positions from shared memory to local memory (Fig. 5). Finally, the linked-cell lists for each subdomain are renewed due to the migration of atoms from one subcell to another.

To make our procedure more instructive, we list four steps that are essential to this approach:

1. Update all interior linked-cell lists.
2. Copy data from interior border subcells into shared memory.
3. Update data in exterior border subcells by copying data from shared memory into private memory.
4. Update linked-cell lists of interior and exterior border subcells.

Note that the linked-cell lists of the interior border subcells need to be renewed twice within this approach.

4 PERFORMANCE RESULTS

Our parallel MD algorithm has been implemented on the BBN TC2000 using the PCP programming paradigm and we have performed some of the largest MD simulations for the longest periods of time to date. Here we present timing comparisons for three-dimensional EAM molecular dynamics simulations containing 4,032, 32,256, and 258,048 atoms. All of our calculations are performed with 64 bit floating-point precision. Table 1 is a summary of the simulation parameters that define our MD simulation. The benchmark calculations are performed for 20 MD time-steps, though we have performed simulations for as long as several million time-steps. The timings from the first 10 time-steps are discarded and the last 10 time-steps are averaged to obtain CPU times for one MD time-step.

On the BBN TC2000, the performance of the startup of a job is severely degraded by the thrashing of the virtual memory system as page tables are constructed for each processor. This effect has been observed on other parallel systems implementing demand paged virtual memory and seems to be inherent in such systems. Although the startup cost does not affect long problems that run for many thousands of time-steps, the chaotic timing effects that arise from the virtual memory system during startup are very inconvenient when attempting to time portions of an application.

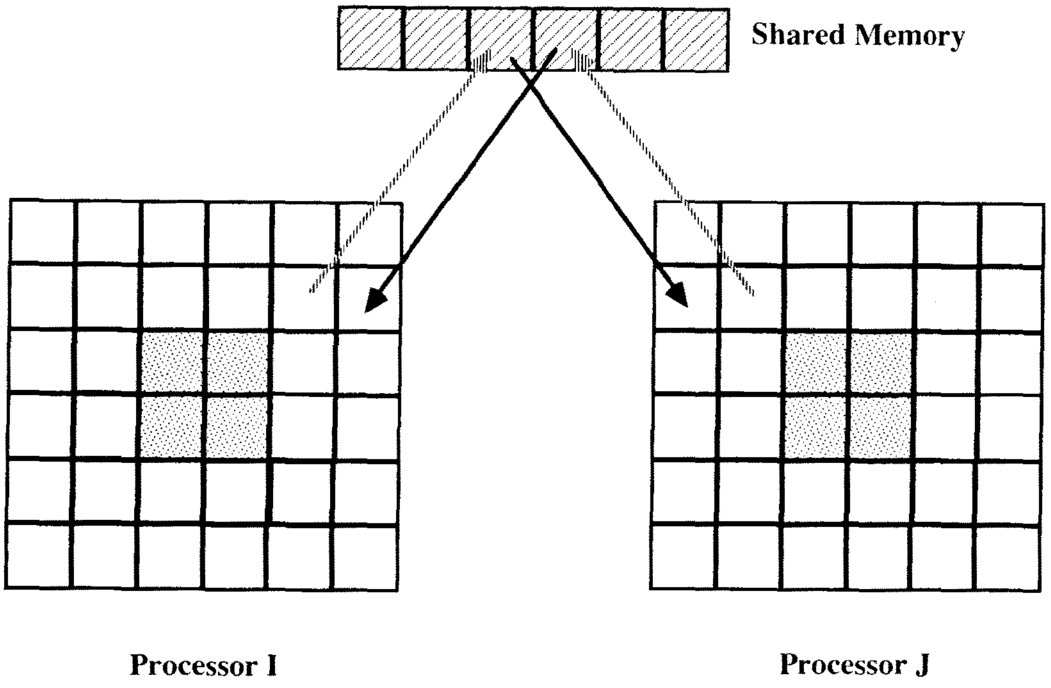

FIGURE 5 The data communication through the shared memory. The data structure within each processor is updated in a two-step manner. The data from the interior border subcells are copied to the shared memory. The updated data from the shared memory belonging to the exterior border subcells are then copied to each processor.

Shown in Tables 2 and 3 are timings for the code executing on the TC2000 using the linked-cell domain decomposition scheme described in this paper. The present code executing on single processor uses memory local to the processor only—there is no shared memory overhead. The share memory is used for interprocessor communication only. There are about eight atoms per subcell and each processor has a private copy of all data that define the calculation. The `celler` and `io` routines are the only routines that operate directly on shared data. From Table 3, we observe that the timings scale better than linear with system size. The ratio of the time spent doing communication to the time spent doing computational work is decreasing with increasing system size.

The main goal of this research was to adapt the domain decomposition scheme into the linked-cell MD code [10] which extensively utilized the shared memory during the MD simulation. although computationally intensive tasks were performed on local data only. It is therefore interesting to compare the timings for these two codes. In Table 4, we show the timing comparisons of these

Table 1. Summary of Physical Variables in Our MD Simulation Presented in this Paper*

Parameter	Typical Value
Desired dimensionless temperature ($T_{desired}$)	0.5 (ε)
Density of the system	0.9421 (σ^{-3})
Relaxation time for ζ	0.10
Time-step in reduced units (Δt)	0.01 (τ)

* The EAM potential parameters were taken from Holian et al. [19]. The simulation state is a hot solid near the melting point.

Table 2. CPU Times (Seconds) to Simulate One MD Time-Step of a Three-Dimensional Material Containing 32,256 Atoms on the BBN TC2000 (The Entire MD Simulation Box is Divided into 4,096 Subcells)

Processors	1	8	64
Total	287.04	33.58	3.76
force	280.22	31.84	3.15
kinetic	0.96	0.13	0.03
celler	4.53	1.25	0.52
ion + others	1.33	0.36	0.06

Table 3. Total CPU Times (Seconds) to Simulate One MD Time-Step of a Three-Dimensional Material Containing 4,032, 32,256, and 258,048 Atoms on the BBN TC2000

Processors	1	8	64
4,032	25.56	3.29	0.56
32,256	287.04	33.58	3.76
258,048	2367.296	209.35	28.22

two codes running simulations of 4,032, 32,256, and 258,048 atoms on 64 processors. We obtain an average of threefold speed-up by using the domain decomposition scheme. Most of this performance increase is attributable to the increase in performance of the `force` routine. Foremost, we have decreased the computational work by a factor of two. Furthermore, we have minimized the interprocessor communication by confining the computational task to local memory.

Figure 6 is a plot of the parallel efficiency of various routines in our code simulating the motion of 32,256 atoms. The parallel efficiency (η) is defined to be the time to execute on one processor (t_1) divided by the time to execute on n processors (t_n) divided by the total number of processors. That is:

$$\eta = \frac{t_1}{t_n x n} \tag{7}$$

The most inefficient routine in our code is the `celler` routine where border subcell information is copied to/from shared memory. The second-most inefficient section of the code is the **io** routine, which we performed once at every time-step in the timings presented here. In a production run,

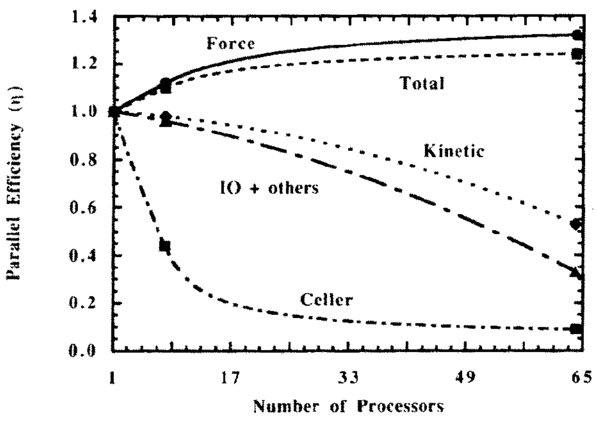

FIGURE 6 The parallel efficiency (η) of the various routines in our MD code simulating 32,256 atoms in three-dimensional EAM. The parallel efficiency (η) is defined to be the time to execute on one processor divided by the time to execute on n processors divided by the total number of processors.

io is executed every ~100 time-steps and the effect on the overall performance is small.

The overall performance is somewhat surprising, providing an efficiency that is greater than 1 and reaching a plateau of 1.2 at around 64 processors. Parallel efficiencies of greater than 1 are unusual, but have been documented before on multiprocessors employing cache equipped microprocessors as their basic computational elements. There are essentially two places where this effect can arise, the first is in the cache used to store virtual to physical address translations and the second is in the cache used to store previously referenced data. The BBN TC2000 is composed of processors possessing individual 16 Kbyte data caches, and memory management units capable of mapping 564K byte pages without TLB misses. As the number of processors is scaled, the total available cache memory scales as well. For a 32K atom problem size, the heavily used portion of the data set begins to fit in the available cache memory as the processor count reaches 64 and no further speed increase results from the cache effect. This effect is important for all currently available microprocessors, but these microprocessors lack hardware monitors that would allow us to pin down the source of the efficiency improvement in a more quantitative manner. We expect that future microprocessors will possess hardware monitors that are capable of counting cache misses and these will be very useful when tracking down anomolous performance results.

Table 4. Total CPU Times (Seconds) and Speed-Ups to Simulate One MD Time-Step of a Three-Dimensional EAM Material Containing 4,032, 32,256, and 258,048 Atoms on the BBN TC2000 of 64 Processors: The Timing Results Are Based on the Same Initial Conditions and Physical Parameters for Both MD Codes and Obtain Identical Physical Results

Number of Atoms	4,032	32,256	258,048
Old	1.62	11.89	93.56
New	0.56	3.74	28.22
Speed-up	2.89	3.18	3.32

5 SUMMARY

In this paper we have described a parallel implementation of our MD code using the linked-cell list method in conjunction with the domain decomposition approach on the BBN TC2000. Instead of using a message-passing programming model for the interprocessor communication, we have employed the available shared memory on the TC2000 as a hub for temporary data storage in the border subcells. We minimize interprocessor communication by confining the computational task to local memory. Most of our performance increase is attributed to the increase in performance of the `force` routine. Our performance results have demonstrated that high parallel efficiency may be obtained on the BBN TC2000 with the PCP programming model. We have obtained roughly three times the performance of the original parallel version of the code on the BBN TC2000 that did not use the domain decomposition technique. Using the domain decomposition technique on 64 processors of the BBN TC2000, we have exceeded 6 times the performance of a partially vectorized version of the algorithm on a single Cray XMP processor. The partially vectorized version generates neighbor-lists from the linked-lists, similar to the algorithm of Grest *et al.* [8]. It is clear that, for large-scale molecular dynamics algorithms with short-range forces, the massively parallel systems compete very favorably with conventional vector supercomputing technology.

ACKNOWLEDGMENTS

This work was performed under the auspices of the U.S. Department of Energy by the Lawrence Livermore National Laboratory under contract No. W-7405-ENG-48.

REFERENCES

[1] D. W. Heermann, *Computer Simulation Methods in Theoretical Physics* (2nd ed.). Berlin: Springer-Verlag, 1990.

[2] W. G. Hoover, A. J. DeGroot, C. G. Hoover, I. F. Stowers, T. Kawai, B. L. Holian, T. Boku, S. Ihara, and J. Belak, "Large-scale elastic-plastic indentation simulations via non-equilibrium molecular dynamics," *Phys. Rev. A*, vol. 42, p. 5844, 1991.

[3] M. S. Daw and M. I. Baskes, "Embedded-atom method: derivation and application to impurities, surfaces, and other defects in metals," *Phys. Rev. B*, vol. 29, p. 6443, 1984.

[4] J. Tersoff, "Modeling solid-state chemistry: interatomic potentials for multicomponent systems," *Phys. Rev. B*, vol. 39, p. 5566, 1989 (and references therein).

[5] L. Verlet, "Computer 'experiments' on classical fluids. I. Thermodynamic properties of Lennard-Jones molecules," *Phys. Rev.*, vol. 159, p. 98, 1967.

[6] D. Fincham and B. J. Ralston, "Molecular dynamics simulation using the Cray-1 vector processing computer," *Comput. Phys. Commun.*, vol. 23, p. 127, 1981.

[7] D. E. Knuth, *The Art of Computer Programming: Volume 1—Fundamental Algorithms.* Reading, MA: Addison-Wesley, 1968.

[8] G. S. Grest, B. Dunweg, and K. Kremer, "Vectorized link cell FORTRAN code for molecular dynamics simulations for a large number of particles," *Comput. Phys. Commun.*, vol. 55, p. 269, 1989.

[9] D. C. Rapaport, "Large-scale molecular dynamics simulation using vector and parallel computers," *Comput. Phys. Rep.*, vol. 9, p. 1, 1988.

[10] J. Belak, *The 1991 MPCI Yearly Report: The Attack of the Killer Micros.* Livermore, CA: Lawrence Livermore National Laboratory UCRL-ID-107022, 1991, p. 219.

[11] A. J. DeGroot, W. G. Hoover, and C. G. Hoover, *The 1991 MPCI Yearly Report: The Attack of the Killer Micros.* Livermore, CA: Lawrence Livermore National Laboratory UCRI-ID-107022, 1991, p. 211.

[12] W. Smith, "Molecular dynamics on hypercube parallel computers," *Comput. Phys. Commun.*, vol. 62, p. 229, 1991.

[13] D. C. Rapaport, "Multi-million particle molecular dynamics II, design considerations for distributed processing," *Comput. Phys. Commun.*, vol. 62, p. 217, 1991.

[14] J. Belak and I. F. Stowers, *Macroscopic and Microscopic Processes*, NATO ASI Series, Series E, Applied Science, Boston, MA: Kluwer Academic, 1992, p. 511.

[15] J. Belak and I. F. Stowers, *Proceedings of the International Conference on Metallurgy Coatings and Thin Films*, 1992, in press.

[16] S. Nosè, "A unified formulation of the constant temperature molecular dynamics methods," *J. Chem. Phys.*, vol. 81, p. 511, 1984.

[17] W. G. Hoover, "Canonical dynamics: equilibrium phase space distribution," *Phys. Rev. A*, vol. 31, p. 1695, 1985.

[18] E. D. Brooks, B. Gorda, and K. Warren, "*The parallel C preprocessor,*" *Scientific Programming*, vol. 1, p. 79, 1992.

[19] B. L. Holian, A. F. Voter, N. J. Wagner, R. J. Ravelo, S. P. Chen, W. G. Hoover, C. G. Hoover, J. E. Hammerberg, and T. D. Dontje, "Effects of pairwise versus many-body forces on high-stress plastic deformation," *Phys. Rev. A*, vol. 43, p. 2655, 1991.