in Materials Theory and Modelling, edited by J. Broughton, P. Bristowe, and J. Newsam, (MRS Proceedings 291, Pittsburgh, PA, 1993), p. 37

PARALLEL MOLECULAR DYNAMICS WITH THE EMBEDDED ATOM METHOD

STEVEN J. PLIMPTON and BRUCE A. HENDRICKSON
Sandia National Laboratories, Albuquerque, NM 87185

ABSTRACT

Parallel computing offers new capabilities for using molecular dynamics (MD) to simulate larger numbers of atoms and longer time scales. In this paper we discuss two methods we have used to implement the embedded atom method (EAM) formalism for molecular dynamics on multiple-instruction/multiple-data (MIMD) parallel computers. The first method (atom-decomposition) is simple and suitable for small numbers of atoms. The second method (force-decomposition) is new and is particularly appropriate for the EAM because all the computations are between pairs of atoms. Both methods have the advantage of not requiring any geometric information about the physical domain being simulated. We present timing results for the two parallel methods on a benchmark EAM problem and briefly indicate how the methods can be used in other kinds of materials MD simulations.

INTRODUCTION

Molecular dynamics (MD) simulation is a commonly used tool in the solid-state physics and materials community for modeling solids and liquids at the atomic level. Each atom in the simulation is treated as a point mass and Newton's equations of motion are integrated to track the motion of each atom. All the physics of the model is contained in an energy functional for the system from which the forces on each atom can be computed. Thermodynamic and transport properties as well as general structural and high-temperature properties of materials can be computed from time averaging various quantities over the ensemble of atoms.

Although the length scale (Å) and time scale (femtosecond timesteps) of MD simulations make them computationally intensive, they have the desirable property that for models of most materials, the computational work scales as N, where N is the number of atoms simulated. This is due to charge-screening effects which limit the range of Coulombic forces meaning each atom interacts only with a small (roughly constant) number of surrounding atoms. Thus in practice, quite large simulations of tens of thousands of atoms can be performed.

For *fcc* and other close-packed metals, the embedded atom method (EAM) [1,2] is a popular choice for the energy functional in MD simulations. It overcomes the volume-dependent limitation of pair-potentials by adding a term for the energy to embed an atom in the background electron density of its neighbors. The EAM has proven particularly good at modeling bulk and defect properties (energy, structure) of metals and metal alloys. For example, many MD and Monte Carlo simulations of surfaces and grain boundaries have been performed using the EAM to model such phenomena as crack growth [1,3], surface reconstruction [4], and grain boundary structure [5] and diffusivity [6]. Computationally the EAM is attractive because it is short-range and, as discussed in the next section, can be computed by summing over pairwise interactions.

In this paper we describe two methods for implementing the EAM for MD on parallel computers, the second of which, we believe, has not been previously proposed as a way of organizing MD computations for parallelism. Optimally, using P processors in a short-range MD model would yield a simulation whose run time scales as N/P. However, this is difficult to achieve in practice, particularly for irregular simulation domains and for small N/P ratios. One goal of this work was to develop a method that would allow even small N simulations to be run quickly in parallel. A second goal was to develop a robust method which would be fast for large

N problems yet not require special coding to exploit the simulation geometry for a specific simulation. The two parallel methods we will describe meet these criteria.

In the next section we highlight the computational steps required for implementing the EAM in a MD simulation. In the following section we detail the two parallel algorithms [7] with a focus on the additional steps necessary for the EAM computations. In the last sections we demonstrate the speed of the algorithms as implemented in an EAM code we use and briefly discuss how the parallel methods can be used in other kinds of MD simulations.

COMPUTATIONAL REQUIREMENTS FOR THE EAM

In the EAM formalism the energy E of atom i is written as

$$E_i = \sum_j \phi(r_{ij}) + F_{embed}\left(\sum_j \rho(r_{ij})\right) \tag{1}$$

where ϕ is a pair-potential interaction between atoms i and j and F is an embedding function, namely the energy to embed atom i in the electron density ρ provided by its surrounding j atoms. The ϕ, ρ, and F functions are analytic expressions with coefficients fit to various experimentally determined quantities [2]. All the functions have a finite range; both summations in equation (1) are over the few dozens of atoms j that surround atom i.

The equation for the force $\vec{F}$ on atom i is the derivative of equation (1) and is given by

$$\vec{F}_i = -\sum_j [\phi'_{ij} + F_i'\rho'_{ij} + F_j'\rho'_{ji}]\vec{r}_{ij} \tag{2}$$

where F_i' is the derivative of atom i's embedding function F, and ρ'_{ij} is the derivative at atom i of the electron density due to atom j. The key computational point is that to calculate the last term one must know F' at each of the j neighbors of atom i. This means the EAM forces are computed in two stages. First the ρ term in equation (1) is summed and embedding functions (and derivatives) are computed for all atoms. Then equation (2) is computed using the F' values. To compute these equations in parallel means that extra inter-processor communication must be done between the two stages. In the first stage, the processor that owns atom i will compute its F and F' values. Then it must be told the F' values for all the j neighbors (owned by different processors) surrounding its atoms so that it can compute equation (2). The parallel algorithms described in the next section perform this extra communication.

PARALLEL ALGORITHMS

There are two basic ideas for performing MD simulations in parallel [7]. When simulating N atoms on P processors, the first idea is to assign N/P atoms to each processor for the duration of the simulation. We call this an "*atom decomposition*" of the workload. Each processor computes forces on only its atoms and updates their positions. At each timestep an all-to-all broadcast is performed so that each processor acquires the positions of all N atoms. This information allows it to compute the ϕ and ρ terms in equations (1) and (2) for its atoms. An all-to-all broadcast of the embedding function F' is also performed between the two stages described in the previous section. The communication for the all-to-all broadcasts can be performed in $\log_2(P)$ steps on a parallel machine in a time proportional to the total amount (N) of information being exchanged [8]. Since this cost is independent of P, it begins to dominate the parallel algorithm for most problems once $P \geq 64$ processors. Nonetheless, the simplicity of the algorithm makes it a good choice for small N on a limited number of processors; it is a way to quickly extract a limited amount of parallelism from a small problem.

The second approach is to exploit the locality of the force interactions by assigning each processor to a small sub-block of the simulation domain - a "*spatial decomposition*" of the workload. Each processor computes forces on the atoms in its block and tracks atoms as they enter and exit its sub-domain. Because the range of the potentials is finite, each processor only need communicate with a few surrounding blocks at each timestep to acquire the necessary information. Extra communication is necessary to move atoms from one block to another as they move through the simulation domain. In the EAM formalism, extra communication is also needed to exchange nearby F' values at each timestep. Spatial-decomposition algorithms are optimal for large N since the communication required scales as the surface-to-volume ratio of each box, $(N/P)^{2/3}$ in 3-d. This means the method is dominated by the N/P scaling of the computation, as desired. However, spatial-decomposition methods are (1) more complex to code efficiently, (2) not as effective when the ratio N/P is small (because each processor's box is small compared to the cutoff length of the potential), and most importantly, (3) require each box (processor) contain roughly the same number of atoms to balance the computational load. The latter point is a problem for physical domains that are irregular or for simulations where the atoms do not fill the domain uniformly. For example, we have been simulating force microscopy tips contacting metal surfaces for adhesion studies. A spatial decomposition of the atoms via a regular 3-d meshing of the simulation domain would cause many processors to be idle since boxes near the tip would contain no atoms.

We propose here a new parallel MD algorithm, particularly well-suited for the EAM computation, which we call a "*force decomposition*" of the workload. It overcomes the drawback of the spatial-decomposition method just mentioned in that it does not require geometric information about the simulation domain to run well. Its run time scales as $N/\sqrt{P}$ which is an improvement on the atom-decomposition method, though not as good as the optimal N/P scaling of the spatial-decomposition method. In practice, however, we have found it to be faster than spatial methods for N up to 10,000 atoms and competitive with them even for N as large as 100,000 atoms.

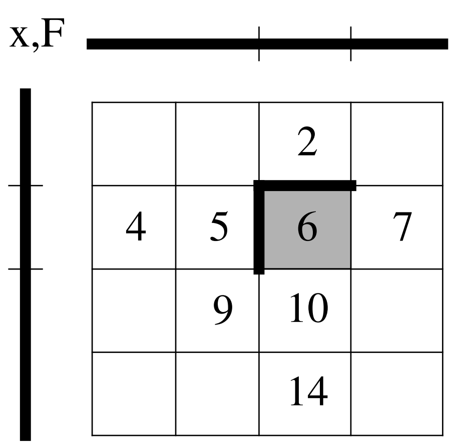

Figure 1: Assignment of processors to the NxN force matrix in the force-decomposition algorithm. Processor 6 owns one block of the matrix and only need store two pieces of the position (x) and embedding (F) vectors, each of length $N/\sqrt{P}$, to compute its matrix elements. The other processors it will communicate with (as discussed in the text) are also numbered.

In the force-decomposition method each processor is assigned N/P atoms for the duration of the simulation, the same as in the atom-decomposition method. The quantities that need to be computed in equations (1) and (2) for all N atoms can be described by a NxN force matrix. Element *ij* of the force matrix is the ϕ, ρ, or force interaction between atom *i* and atom *j*. Thus the total force on atom i is the sum of all elements in row i of the matrix. Note that the matrix is sparse (short-range forces) and symmetric (Newton's 3rd law). Rather than have a processor compute all forces on its N/P atoms (compute N/P rows of the matrix) as we did in the atom-decomposition method, we make an important change. We assign each processor to a block of the force matrix as shown in Figure 1. If the atoms are ordered randomly, each block will have roughly the same number of non-zero elements to compute (random sparsity). Also note that in order for a processor to compute its matrix elements it does not need to know the positions (or embedding functions) of all N atoms (as it did in the atom-decomposition method) - only those of two sub-groups of atoms each of size $N/\sqrt{P}$, as indicated in the figure. We now outline the EAM MD computation at each timestep using the force matrix decomposition just described. For simplicity we assume P=16 processors as in the figure. We assume that at the beginning of the timestep each processor has just updated the positions of its N/P atoms. Each processor then does the following:

- Acquire the $N/\sqrt{P}$ positions of atoms in its row of the matrix (processor 6 communicates with 4,5,7).
- Transpose (exchange) its N/P positions with the appropriate processor in the transposed matrix location (6 exchanges with 9).
- Acquire the $N/\sqrt{P}$ positions of atoms in its column (6 communicates with 2,10,11).
- Compute the electron density (ρ) matrix elements in its block.
- Sum the ρ values across the matrix row (6 sums with 4,5,7) for its N/P atoms.
- Compute the embedding functions (F and F') for its N/P atoms using the summed ρ values.
- Repeat the first 3 steps with F' instead of the atom positions to acquire all the F' values for the $N/\sqrt{P}$ atoms in its rows and columns of the matrix.
- Compute the equation (2) force terms in its block using the F' values.
- Sum the forces across the row (6 sums with 4,5,7) for its N/P atoms and update their positions.

While there are many steps, each of them is a simple operation to program and perform on a parallel machine. More importantly, all of the communication is between small groups of $\sqrt{P}$ processors and requires information exchanges of length $N/\sqrt{P}$ or less (of length N/P for the transposed quantities). This is in contrast to the atom-decomposition method where all P processors exchange vectors of length N. This means that the scaling of the force-decomposition algorithm is $N/\sqrt{P}$ (vs. N for atom-decomposition, vs. N/P for spatial-decomposition). Thus more processors can be effectively used to solve a particular problem than with the atom-decomposition method. We also note that, as outlined above, the method does not take advantage of Newton's 3rd law to halve the necessary computation. This can be done by "checker-boarding" the force matrix in figure 1, so that elements above the matrix diagonal are 0 if (i+j) is odd and elements below the diagonal are zero if (i+j) is even. Each pair interaction is then computed only once in the matrix (by a single processor). Additional communication steps (similar to those above) are then necessary to sum computed values within columns of the matrix and to transpose the results to the rows before the row summations are done. Timing results for a parallel EAM code which implements the checker-boarding scheme are given in the next section.

RESULTS

We have implemented both an atom- and force-decomposition algorithm in a parallel EAM modeling code we use. The parallel code is based on the vector code DYNAMO written by Stephen Foiles and Murray Daw at Sandia Livermore. The effectiveness of these parallel methods is shown in timings for a Cu benchmark problem of variable size (N) in Figure 2. The EAM functionals for Cu are truncated at 5Å so that each Cu atom interacts with about 50 neighbors (80 atoms in an extended Verlet neighbor list). Since different researcher's implementations of the EAM differ in the analytic forms chosen for ϕ, ρ, and F and in the interpolation methods used to compute them, we compare here to the Foiles and Daw version which vectorizes well and is optimized for the CRAY. The parallel code uses the same functional forms and interpolation methods. The CRAY timings (single Y-MP processor) are shown in the figure for various sized problems (N) along with timings on a 1024-processor nCUBE 2 for both the atom- and force-decomposition method. The nCUBE 2 is a distributed memory parallel computer in the same price range as a single Y-MP processor.

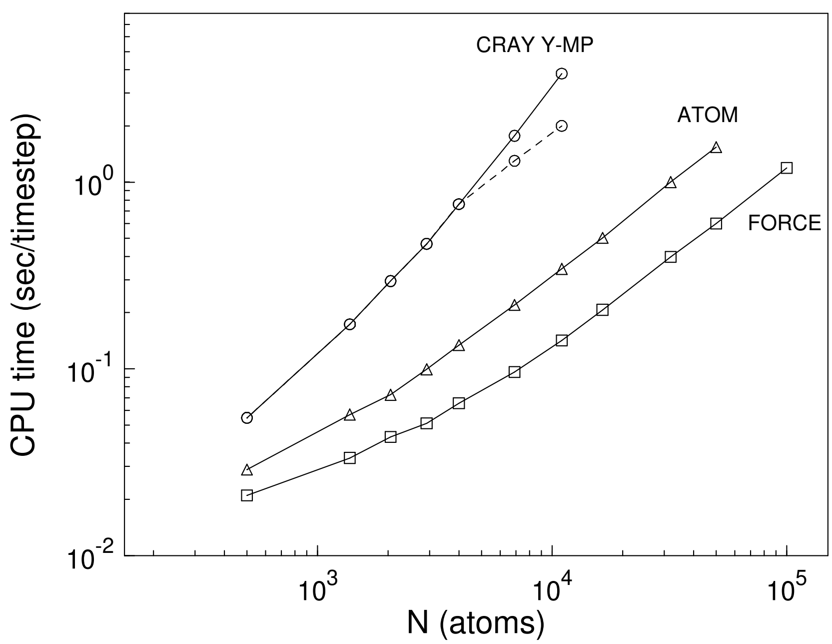

Figure 2: CPU time (per timestep) for an EAM benchmark simulation of Cu as a function of the size (N) of the problem. The CRAY timings are for a single Y-MP processor. The other 2 curves are timings of the atom- and force-decomposition algorithms on a 1024-processor nCUBE 2 parallel computer.

The dotted line for the CRAY is an estimate of the vector code's timing if a neighboring method (e.g. link-cell) more appropriate for large N were implemented; this has not been done previously since problems with N > 10,000 atoms are rarely run with the CRAY version of DYNAMO. The timing results in the figure show the parallel machine to be 5-15x faster than a single Y-MP processor across a wide range of N. The force-decomposition method is about 50% efficient for $N \geq 10^4$ atoms, meaning that it runs within a factor of two of perfect speed-up. Our experience with pair potentials [7] has been that the more complex spatial-decomposition methods are not faster than the force-decomposition method until about $N \sim 10^4$ atoms and only achieve near-optimal efficiency for hundreds of thousands of atoms. Thus the force-decomposition method is the fastest choice for EAM simulations of up to ten-thousand atoms and competitive with the optimal methods even for N in the tens-of-thousands of atoms regime.

CONCLUSIONS

We have discussed two algorithms suitable for parallel implementation of MD simulations using the EAM. Their chief advantage is that they require no geometric (spatial) information about the simulation domain in order to partition the computational work equally among the processors. Both are relatively simple to implement and the force-decomposition method in particular achieves reasonable parallel efficiencies (50%) across a wide range of problem sizes. It has allowed us to simulate small to large ensembles of atoms in a fraction of the time it would take on a conventional vector supercomputer. We have used the methods in recent months to simulate grain boundary diffusion [earlier related work in 9,10], adhesion between metal surfaces and force microscopy tips, and to study the liquid-solid interface in *fcc* metals (with Mike Uttormark of the University of Wisconsin-Madison). All of the kinds of auxiliary computations one typically performs as a MD simulation progresses (thermodynamics, transport coefficients, energy minimization, temperature control, etc.) are easily implemented in parallel as add-ons to the methods we have described.

Lastly, we also emphasize that the parallel methods themselves are not useful only for MD using the EAM. They can be straightforwardly applied to pair-potential models [7] and the atom-decomposition method works well for three-body potentials such as those for Si and silica [11] or higher-order many-body computations. We have also recently extended the force-decomposition method to covalently bonded systems [12] such as polymers where one must do additional work to compute 2-, 3-, and 4-body (bonded) interactions. Additionally, we note that these algorithms can be easily coded for (and ported between) any distributed memory parallel machines which perform message passing to exchange information between processors (MIMD parallel machines). Examples include the nCUBE 2, Intel (Gamma, Delta, and Paragon) machines, and the CM-5 (in message-passing mode).

ACKNOWLEDGMENTS

The authors thank Grant Heffelfinger at Sandia for helping with this work. They also thank Stephen Foiles and Murray Daw at Sandia Livermore for many useful discussions concerning the EAM and its implementation in their vector code DYNAMO. This work was performed at Sandia National Laboratories, operated for the U.S. Department of Energy under contract number DE-AC04-76DP00789.

REFERENCES

[1] M. S. Daw and M. I. Baskes, Phys. Rev. B **29**, 6443 (1984).
[2] S. M. Foiles, M. I. Baskes, and M. S. Daw, Phys. Rev. B **33**, 7983 (1986).
[3] M. I. Baskes, M. S. Daw, B. W. Dodson, and S. M. Foiles, MRS Bulletin, **13**, 28 (1988).
[4] S. M. Foiles, Surf. Sci. **191**, L779 (1987).
[5] K. L. Merkle, D. Wolf, MRS Bulletin, **15**, 42 (1990).
[6] M. Nomura, S. Lee, and J.B. Adams, J. Mater. Res. **6**, 1 (1991).
[7] S. J. Plimpton and B. A. Hendrickson, Sandia Report S91-1144J, available from Sandia National Labs, Albuquerque, NM, submitted for publication.
[8] G. Fox, M. Johnson, et. al., Solving Problems on Concurrent Processors, Vol 1 (Prentice-Hall, New Jersey, 1988), p. 407.
[9] S. J. Plimpton and E. D. Wolf, Phys. Rev. B **41**, 2712 (1990).
[10] S. J. Plimpton, in Atomic Scale Calculations of Structure in Materials, edited by M. S. Daw and M. A. Schluter (Mater. Res. Soc. Proc. **193**, Pittsburgh, PA. 1990) p. 333.
[11] M. H. Grabow, C. R. Kurkjian, D. Inniss, S. J. Plimpton, to be presented at Am. Ceramic Soc. Mtg., Cincinnati, OH, April 1993.
[12] S. J. Plimpton, B. A. Hendrickson, and G. S. Heffelfinger, to appear in proceedings of Sixth SIAM Conference on Parallel Processing for Scientific Computing, April 1993.