CILAMCE

Ouro Preto/MG - Brazil

2003

XXIV IBERIAN LATIN-AMERICAN CONGRESS ON COMPUTATIONAL METHODS IN ENGINEERING

A PARALLEL IMPLEMENTATION FOR A DRIVEN MOLECULAR DYNAMICS ALGORITHM

Stephan Stephany
Airam Jonatas Preto
Laboratório Associado de Computação e Matemática Aplicada (LAC/INPE)
[stephan,airam]@lac.inpe.br
Enzo Granato
Laboratório Associado de Sensores e Materiais (LAS/INPE)
enzo@las.inpe.br
José Alexandre Junqueira Ribeiro
Programa de Pós-graduação em Computação Aplicada (CAP/INPE)
jajribeiro@ieg.com.br
Instituto Nacional de Pesquisas Espaciais (INPE)
Caixa Postal 515, CEP 12245-970 São José dos Campos, SP - Brasil

Abstract. *Molecular dynamics algorithms are used in computational experiments simulating the properties of liquids, solids and molecules. The n atoms or molecules in the simulation are treated as point masses and Newton's equations are integrated to compute their motion at each time step. Parallel implementations of molecular dynamics algorithms use either particle decomposition, force decomposition or space decomposition schemes in order to decompose the domain. We present a parallel implementation for a driven molecular dynamics simulation based on the particle decomposition method. The simulation of an adsorbed monolayer under an external driving force, relevant for sliding friction phenomena of crystalline surfaces, is used to test the parallel algorithm. The algorithm is suitable for both short-range and long-range interactions and can also be converted to a space decomposition method for short-range interactions. The code was parallelized using the MPI communication library. A logical ring is defined in which each processor receives data from its left neighbor and sends data to the right neighbor. This pipeline scheme takes advantage of the Newton 3rd law and drastically reduces the number of redundant calculations. This scheme yields an efficient parallelization and results show good speed-up's running on a standard distributed memory parallel machine.*

Keywords: *parallel programming, N-body simulation, molecular dynamics, MPI*

1. INTRODUCTION

Molecular dynamics algorithms are often used in computational experiments simulating the properties of liquids, solids and molecules. The n atoms or molecules in the simulation are treated as point masses and Newton's equations are integrated to compute their motion at each time step (Allen, 1993, Rapaport, 1995). If the force between particles is additive and acts along the line between them, this class of problems is also known as N-body problem in computer science (Barnes and Hut, 1986, Singh et al., 1995). However, typical N-body algorithms are designed for long-range forces and, because of their expense, are not commonly used in classical molecular dynamics simulations (Plimpton, 1995).

To implement the molecular dynamics simulation, a potential energy functional is adopted and individual force differential equations are derived for each of the n particles in the system. The equations are then solved numerically, under appropriate boundary conditions, giving the time dependence of the all particle positions and velocities. Usually, the temperature of the system is kept constant on the average by readjusting the velocities such that the average kinetic energy is constant. These simulations methods are important for the study of microscopic and macroscopic properties such as transport coefficients, phase diagrams and structural properties. In particular, recent work on friction properties of crystalline surfaces (Persson, 1998) has employed driven molecular dynamics simulation to study the sliding friction of adsorbed layers under an external driving force (Persson, 1993, Granato and Ying, 2000). In this case, to insure constant temperature, a stochastic method of integrating the dynamical equations, also known as Brownian molecular dynamics (Allen, 1993), was used, which corresponds to coupling the system (adsorbed layer) to a heat bath represented by additional random forces.

Molecular dynamic simulation are not memory intensive as the data structures are restricted to vectors of dimension n that store the position, velocity and eventually other particle properties. However, the processing load is heavy due to the need of calculating interaction forces between pairs of particles and therefore the algorithmic complexity scales as $O(n^2)$. In addition, a large number of time steps is usually required, specially in simulations of nonequilibrium properties by driven molecular dynamics.

A simulation of an adsorbed monolayer under a driving force (Granato and Ying, 2000) is used to test a parallel algorithm for driven molecular dynamics. The n particles are bound to a 2D domain with periodic boundary conditions and constant density. The interaction energy between each pair of particles is given by

$$v(r_{ij}) = \epsilon \left[\left(\frac{\sigma}{r_{ij}} \right)^{12} - 2 \left(\frac{\sigma}{r_{ij}} \right)^{6} \right] , \tag{1}$$

where r_{ij} is the distance between particles i and j, σ is the natural equilibrium distance between particles and ϵ is the strength of the interaction potential. These are parameters that define the Lennard-Jones potential. Because this potential is short ranged, it is sufficient to consider only the interactions between particles that are at a radius less than a cut-off distance, due to its strong exponential decay. Several algorithms take advantage of this simplification. In addition to the interaction with other particles by the above potential, each particle of the adsorbed monolayer also interacts with the substrate by the periodic potential

$$u(\mathbf{r}_i) = U_o[2 - \cos(2\pi x_i/a) - \cos(2\pi y_i/a)] , \tag{2}$$

where a is the lattice spacing of the substrate.

Parallel implementations of molecular dynamics algorithms use either particle decomposition, force decomposition or space decomposition schemes in order to decompose the domain (Plimpton, 1995). This work presents a parallel implementation for the driven molecular dynamics simulation introduced above based on the particle decomposition method. The algorithm is suitable for both short-range and long-range interactions and can also be converted to a space decomposition method for short-range interactions. The code was parallelized using the MPI communication library. A logical ring is defined in which each processor receives data from its left neighbor and sends data to the right neighbor. This pipeline scheme takes advantage of the Newton 3rd law and drastically reduces the number of redundant calculations. Even considering a potential which requires the calculation of the interactions between all pairs of particles, the scheme yields an efficient parallelization. Results show good speed-up's running on a standard distributed memory parallel machine.

2. BROWNIAN MOLECULAR DYNAMICS SIMULATION

In the Brownian dynamics simulation (Allen, 1993), also known as Langevin dynamics simulation, thermal equilibrium at a specified temperature T is insured by a procedure that mimics frequent collisions of the particles of the system with much lighter particles representing a heat bath and described by a random force distribution specified by the given reference temperature T and a damping coefficient parameter η. The dynamics of the n particle system is described by the stochastic equation

$$m\ddot{\mathbf{r}}_\mathbf{i} + m\eta\dot{\mathbf{r}}_i = -\frac{\partial V}{\partial \mathbf{r}_i} - \frac{\partial U}{\partial \mathbf{r}_i} + \mathbf{f}_i + \mathbf{F} \quad , \tag{3}$$

where $\mathbf{r}_i$ is the particle position and $U = \sum_i u(\mathbf{r}_i)$ and $V = \sum_{i \neq j} v(|\mathbf{r}_i - \mathbf{r}_j|)$ are giving by Eqs. (1) and (2). $\mathbf{F}$ is a uniform external force acting on each particle and $\mathbf{f}_i$ is an uncorrelated stochastic force, with zero average, and variance related to the damping coefficient parameter η, the mass of the particles m and the temperature T by the fluctuation-dissipation relation

$$< f_i^\alpha(t) f_j^\beta(t') >= 2\eta mkT\delta(t-t')\delta_{\alpha,\beta}\delta_{i,j} \quad , \tag{4}$$

where α, β represent the vector components. Typical quantities of interest in these simulations are the equilibrium averaged potential energy of the system e_p, defined as

$$e_p =< \frac{1}{2}\sum_{i \neq j} v(\mathbf{r}_{ij}) + \sum_i u(\mathbf{r}_i) > \quad \text{for} \quad F = 0 \quad , \tag{5}$$

and the center of mass mobility, defined as $\mu = v_{cm}/F$ for small $\mathbf{F}$, where v_{cm} is the center of mass velocity. These quantities have been used to monitor the parallelization scheme.

The dynamical equations are solved numerically in discrete time steps using a simple leap-frog type scheme (Allen, 1993). The force between the n particles in each time step, corresponding to the 1st term on the righthand side of Eq. 3, is calculated by adding the forces between all pairs of particles. This requires $O(n^2)$ computations. Considering a parallel implementation of this algorithm, if the n particles are distributed among p processors, $O(p^2)$ communications between processors will be required during each time step of the simulation. If the force between particles falls rapidly with distance, as for the Lennard-Jones potential in Eq. 1, the influence of distant particles becomes negligible. In this case, a *cut-off distance* can be employed to reduce the number of computations.

3. PARALLEL PROGRAMMING

The prevailing trend in the search for high performance is the use of parallel machines due to their good cost effectiveness. Two parallel architectures are usually considered: shared memory and distributed memory machines. In the former class, the multiprocessors, all processors access a unique memory address space and there are scalability constraints. In the latter class, off-the shelf machines called nodes are interconnected by a network composing a multicomputer or cluster. In the case of hundreds or thousands of nodes using a very fast interconnection scheme the multicomputer is called a MPP (massive parallel processors).

The processors of each node access only their local memories and data dependencies between node memories enforce communication by means of routines of a message passing library, like MPI (Message Passing Interface) or PVM (Parallel Virtual Machine). All current world top performance machines (supercomputers) are MPP's (Foster, 1995). Another point is the use of scalar or vector processors in parallel machines. The use of vector processors is restrained to specialized applications as wheather forecasting due to cost issues, but even in this area there is a migration from shared-memory vector-processor nodes to scalar-processor MPP's.

MPI supports the SPMD (single program multiple data) scheme in which each processor/node executes the same subset of instructions in a subdomain of data defined according to its rank (Pacheco, 1996). Data dependencies between processors require calls to the MPI library. On the other hand, a single processor/node may perform tasks like gathering partial results obtained by every node and broadcasting the global result to all other nodes, also by means of MPI calls.

An important issue in parallel programming is to maximize the amount of computation done by each processor and to minimize the amount of communication, due in this case to MPI calls, in order to achieve good performance. This is defined as *granularity*, the amount of processing between processor communication/synchronization. This is particularly important in multicomputers as the communication latency is relatively high. However, the code must be instrumented in order to obtain timing and profiling data that will be used to select the parts of the code that will be parallelized (Stephany et al., 2000).

The speed-up $S_p(n,p)$ is defined as the ratio between the sequential execution time $T_s(n)$ and the parallel execution time $T_p(n,p)$ for p processors and a problem of size n. In this work, n denotes the number of particles. A linear speed-up denotes that processing time was decreased by a factor of p when using p processors and can be thought as a kind of nominal limit.

$$S_p(n,p) = \frac{T_s(n)}{T_p(n,p)} \quad , \qquad\qquad 0 \le S_p(n,p) \le p \tag{6}$$

The efficiency $E_p(n,p)$ is defined as the ratio of the speed-up by p and thus it is 1 for a linear speed-up. Usually, communication penalties cause $E_p(n,p) < 1$ and particularly, a *slowdown* occurs if $E_p(n,p) < 1/p$.

$$E_p(n,p) = \frac{S_p(n,p)}{p} \quad , \qquad\qquad 0 \le E_p(n,p) \le 1 \tag{7}$$

Exceptionally, as data is partioned among processors, cache memory access can be optimized in such way that superlinear speed-up's can be attained, i.e. $S_p(n,p) > p$ and $E_p(n,p) > 1$ (Campos Velho et al., 2002).

4. THE CIRCULAR PIPELINE ALGORITHM

The sequential code was parallelized using the MPI (Message Passing Interface) message passing communication library (Gropp et al., 1999). A particle decomposition scheme was adopted with sets of particles being assigned to each processor. A logical ring is defined in which each processor receives interaction force data from its left neighbor and sends it to the right neighbor taking advantage of the Newton 3rd law to reduce redundant calculations. At every time step each processor calculates interaction forces and exchange part of this data in order to update velocity and position of its particles. The coordinates of all particles are broadcasted at the end of the time step also using a logical ring scheme. Denoting by p the number of processors, the total number of force-interaction calculations in the sequential case is given by the classical formula $[n^2/2 - n/2]$, while the total number of calculations in this implementation, using p processors, is given by

$$\left[\left(\frac{p+1}{p}\right)\frac{n^2}{2} - \frac{n}{2}\right] \tag{8}$$

Therefore, the penalty in the number of calculations for the parallel version is proportional to $[(p+1)/p]$. This penalty scales very well with p. Although a cut-off distance is used in this parallel implementation in face of the Lennard-Jones potential, it could be used for any long range potential, or any N-body problem. On the other hand, it can be further optimized for short range potentials by spatially grouping the particles among processors in addition to the logical grouping. This would require an update of processor particles at a defined interval of time steps.

Denoting by *local forces* the interaction-forces between particles of the same processor p_i and by *remote forces* those between particles of processor p_i and particles of other processors p_j. Each processor computes a set of *local forces* and $(p-1)$ sets of *remote forces*. A possible straightforward parallel algorithm, executed in each processor p_i and using MPI can be expressed by

```
FOR EACH TIMESTEP DO
    CALCULATE LOCAL FORCES F(ii)
    FOR EACH PROCESSOR(j) DO
        CALCULATE REMOTE FORCES F(ij)
    ENDDO
    CALCULATE RESULTANT FORCES
    UPDATE VX(i), VY(i)
    UPDATE X(i), Y(i)
    BROADCAST X(i), Y(i), VX(i), VY(i)
    FOR EACH PROCESSOR(j) DO
        BROADCAST X(j), Y(j), VX(j), VY(j)
    ENDDO
ENDDO
```

In this pseudocode, the n-dimension vectors X and Y denote particle coordinates, while the n-dimension vectors VX and VY, particle velocity components.

The general scheme of the interaction-force logical takes advantage of a logical ring for exchanging forces and another one for positions. Therefore, two circular pipelines are implemented, as follows

```
FOR EACH TIMESTEP DO
    CALCULATE LOCAL FORCES F(ii)
    FOR EACH PROCESSOR(j) DO
        CALCULATE REMOTE FORCES F(ij)
        IF SUITABLE
           SEND F(ij)
           RECEIVE F(ji)
        ENDIF
    ENDDO
    CALCULATE RESULTANT FORCES
    UPDATE VX(i), VY(i)
    UPDATE X(i), Y(i)
    FOR EACH PROCESSOR(j) DO
        SEND X(i), Y(i), VX(i), VY(i)
        RECEIVE X(j), Y(j), VX(j), VY(j)
    ENDDO
ENDDO
```

In the interaction-force logical ring, the “suitable” condition to be tested is related to number of sets of *remote forces* that can be received from other processors making use of the Newton 3rd law. In this scheme, each processor computes a set of *local forces* and $p/2$ sets of *remote forces*. For example, in the case of $p = 10$, instead of computing 9 sets of *remote forces*, each processor computes only 5 sets. In the case $p = 15$, 7 sets, instead of 14. This is expressed by Eq. 8.

5. PERFORMANCE RESULTS

In this work, parallel execution was performed on a distributed memory parallel machine combining a low cost architecture and free software. This cluster is composed by 17 monoprocessed IA-32 scalar nodes running Linux and a Fast Ethernet switch. A Fortran 90 (Metcalf & Reid, 1999) optimizing compiler and the message passing communication libray MPI (Message Passing Interface) were used to parallelize the code. Simulations were performed for up to 28800 particles and the reduction of the processing time for the parallel version in comparison to the sequential one is shown in Table 1 and Fig. 1.

Table 1: Processing times in seconds for 5, 10 and 15 processors and n particles

n	p=5	p=10	p = 15
4050	230.11	103.29	69.80
11250	1887.10	727.37	457.60
16200	3582.44	1497.16	936.20
28800	8323.00	4679.00	2901.70

This table contains the processing times for some simulations depicted in the latter figure and show that this algorithm, executed in this low cost parallel architecture, is suitable for this kind of computational experiments, yielding a a significant reduction of the processing time.

However, as shown in Fig. 2, the overhead caused by the parallelization makes it inefficient for small number of particles. In this particular case, the sequential code runs faster up to 250 particles, but the parallel code speed-up very fast as the number of particles increase.

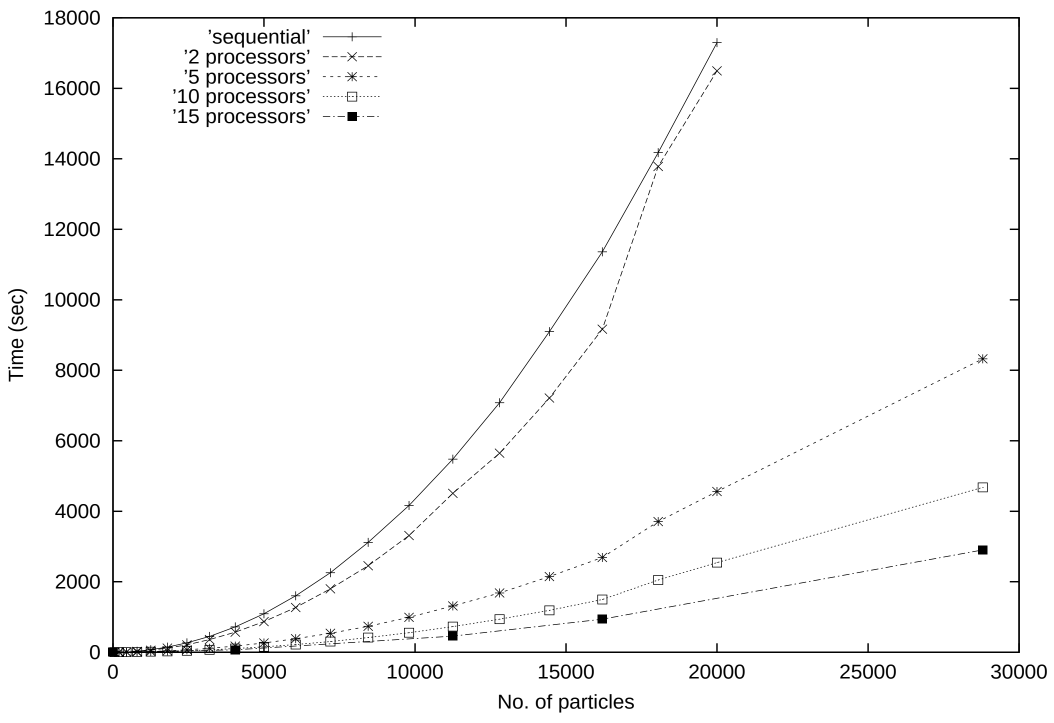

Figure 1: Processing times versus number of particles for 1000 time steps

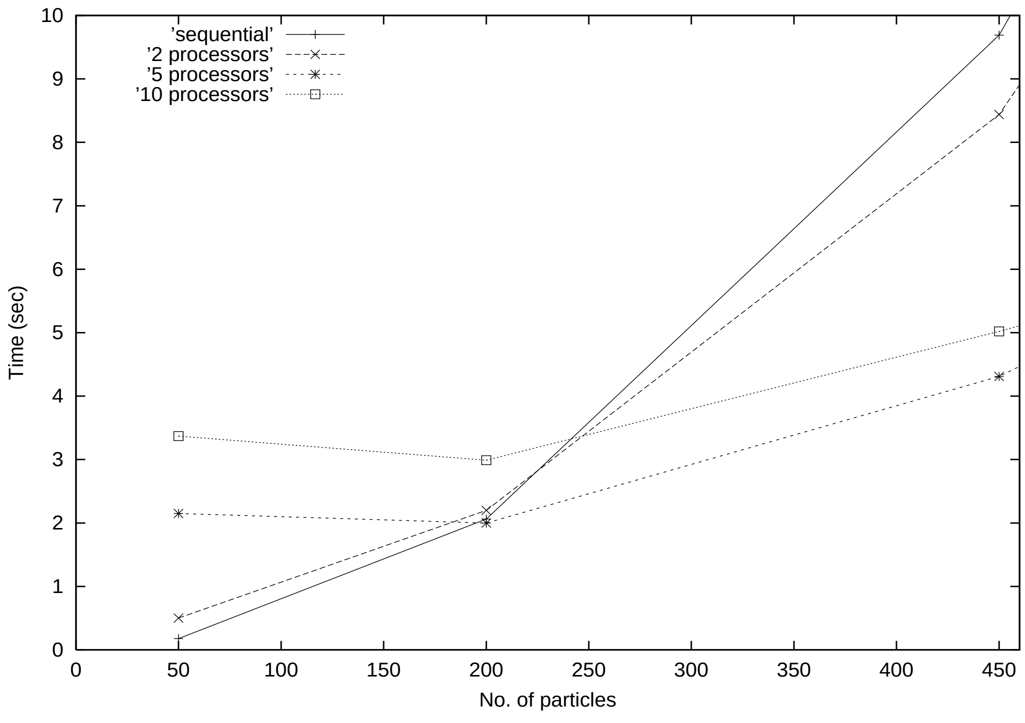

Figure 2: Processing times for small number of particles and 1000 time steps

Table 2 and Fig. 3 shows the speed-up considering two instances of n (9800 and 20000 particles) for different p's (2, 5 and 10). As it would be expected, communication time tends to increase as p increases and this penalty precludes a linear speed-up.

Table 2: Speed-up for 9800 and 20000 particles

n	p=2	p=5	p = 10
9800	1.3	4.2	7.5
20000	1.1	3.8	6.8

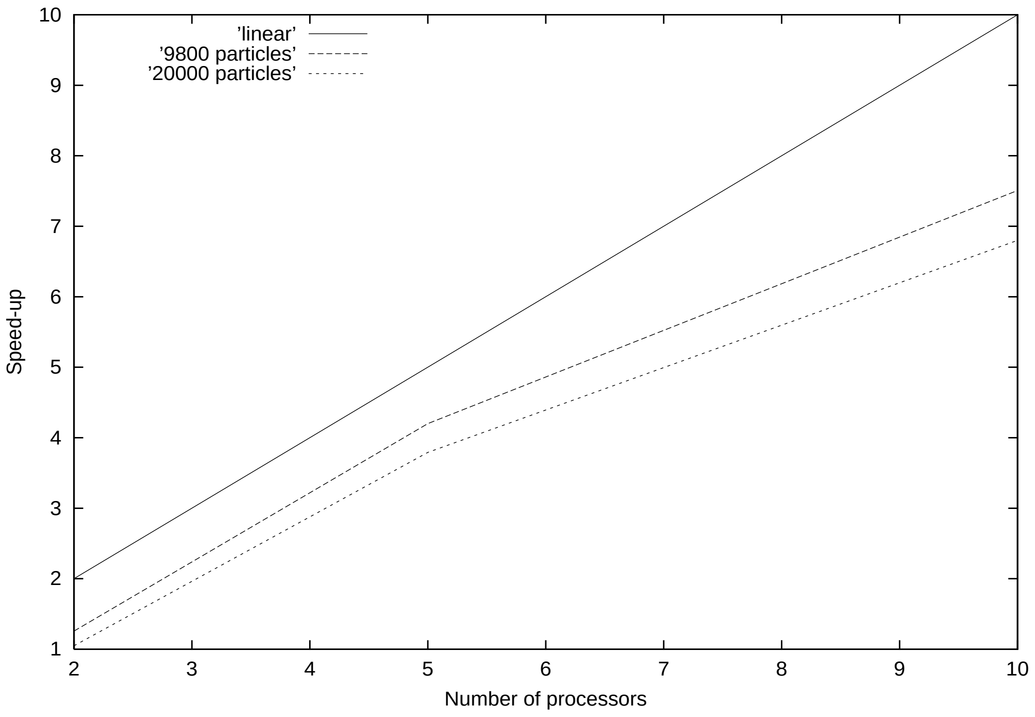

Figure 3: Speed-up for 9800 and 20000 particles

Finally, Fig. 4 show how the total execution time compares to the processing and communication times for the simulations using 10 processors and up to 28800 particles. It should be pointed out that this figure shows the *minimum* processing and *maximum* communication times among the p processors in order to show the overhead due to interprocessor communication.

The instrumentation of the code in order to provide timing information is straightforward, by means of calls to the MPI_WTIME() routine of the MPI library. At each time step, a simple reduction operation, MPI_REDUCE, allows to collect the maximum and minimum processing and communication times from all the processors and to accumulate these partial times for the whole simulation.

Despite the communication penalty, the timing results show that the algorithm scales well since a good granularity is obtained for the considered number of processors p used in the current architecture and the number of particles n of the test cases. Another point is that for this particular simulation, to increase the number of time steps can be more interesting rather than increasing the number of particles n, provided that this quantity be sufficiently large to represent the system of particles involved in the phenomenon.

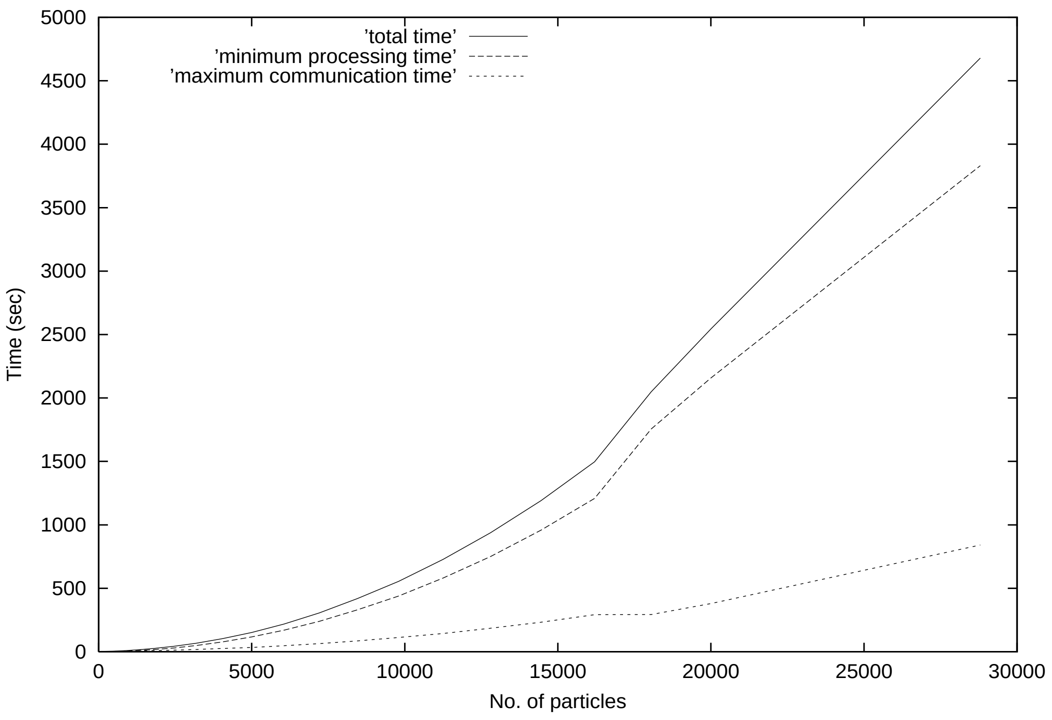

Figure 4: Algorithm granularity for 10 processors

6. FINAL COMMENTS

As an illustration of useful physical information that can be obtained from these simulations, we show in Figure 5 the behavior of the mobility μ as function of temperature T for a system containing 450 particles, using the parallel version of the computer code described above. For temperatures below a critical value $T_m \sim 1.2U_o/k$, the mobility vanishes indicating that the adsorbed layer is pinned by the periodic potential while above T_m the mobility is nonzero and the adsorbed layer is melted.

The insets in Fig. 5 show the equilibrium particle configurations of a portion of the system at temperatures below and above T_m. Grid lines represent the periodic potential of the substrate. These configurations reveal that below T_m, the particles are located at the minima of the periodic potential (grid-lines intersections), forming a structure commensurate with the substrate, corresponding to a solid phase. Above T_m, the structure is disordered as in a liquid phase.

The pinned solid phase and its melting transition at T_m have important consequences for the nonlinear response behavior of the adsorbed layer at large applied forces. Such behavior is particularly relevant for the study of sliding friction phenomena between two crystalline surfaces (Persson, 1993, Persson, 1998, Granato and Ying, 2000).

It should be pointed that the physical results, like the ones described above, for different temperatures, were required to validate the current parallel algorithm. These simulations are very sensitive to numerical errors and using the numerical value of a single quantity for a fixed temperature is not reliable enough, due to the stochastic nature of the dynamics described by Eq. (3).

This work has shown that a cost effective architecture can be successfully employed to perform high performance molecular dynamics simulations using standard software tools. A $O(n^2)$ scaling, circular pipeline scheme was implemented for a molecular dynamics problem. This scaling is still worst than $O(n \log n)$, the Barnes-Hut algorithm (Barnes and Hut, 1986,

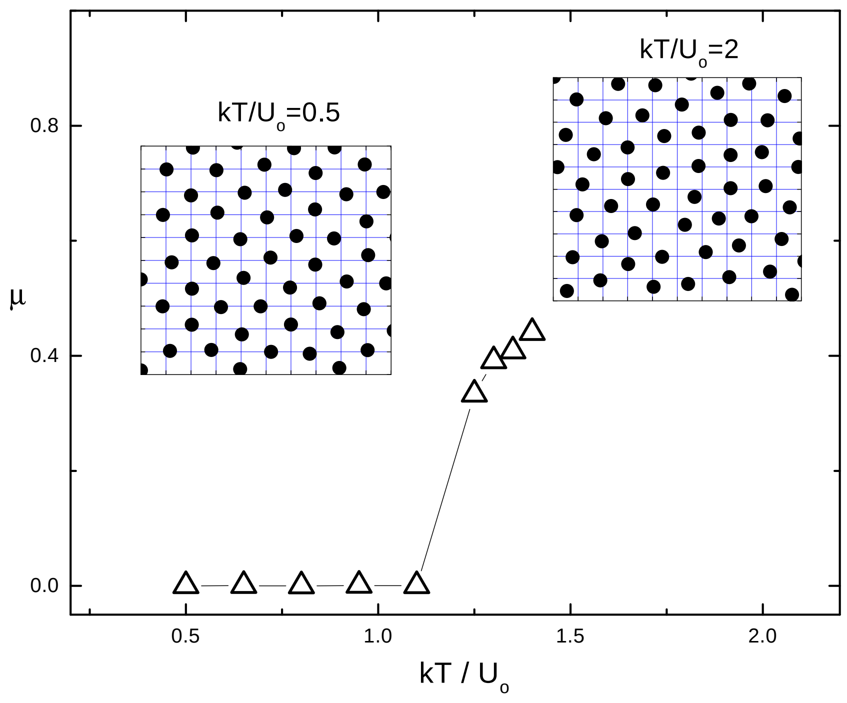

Figure 5: Mobility μ as a function of temperature T for $n = 450$ particles. Insets show particle configurations of a portion of the system for $T < T_m$ (left) and $T > T_m$ (right). Grid lines represent the periodic potential of the substrate.

Singh et al., 1995) scaling. However, the current algorithm can be further upgraded to take advantage of short range potentials, Actually, a standard strategy, the spatial decomposition of the domain, is currently being implemented, in order to decrease the number of interactions that have to be computed at each time step and therefore to improve the speed-up and the efficiency.

Acknowledgements

This work is supported by FAPESP, The State of São Paulo Research Foundation, research project grant 01/03100-9, *Paralelização de Aplicações em Física dos Materiais num Ambiente de Memória Distribuída*. One of the authors (EG) is also supported by FAPESP research project grant 03/00541-0, *Dinâmica e Transições de Fase em Sistemas Periódicos*.

REFERENCES

Allen, M.P., 1993, "Computer Simulation of Liquids", Oxford University Press, New York, USA.

Barnes, J.E., Hut, P.,1986, "A hierarquical O(nlogn) force-calculation algorithm", Nature v. 324, pp. 446-449.

Campos Velho, H. F., Stephany, S., Preto, A. J., Vijaykumar N. L., Nowosad A. G., "A neural network implementation for data assimilation using MPI, in Applications of High-Performance Computing in Engineering VII, eds. C.A. Brebbia, P. Melli and A. Zanasi", pp. 211-220, WIT Press, Southampton, UK.

Foster, I., 1995, “Designing and Building Parallel Programs”, Addison-Wesley, New York, USA.

Granato, E., Ying, S.C., 2000, “Transverse thermal depinning and nonlinear sliding friction of an adsorbed monolayer”, Physical Review Letters v. 85, pp. 5368-5371.

Gropp, W., Lusk, E. & Skkjellum A., 1999, “Using MPI - Portable Parallel Programming with the Message Passing Interface”, The MIT Press, Cambridge, USA.

Metcalf, M. & Reid, J., 1999, “Fortran 90/95 Explained”, 2nd edition, Oxford University Press, Oxford, UK.

Pacheco P., 1996, “Parallel Programming with MPI”, Morgan Kaufmann Publishers, USA.

Persson, B.N.J., 1993, “Theory and simulation of sliding friction”, Physical Review Letters v. 71, pp. 1212-1215.

Persson, B.N.J., 1998, “Sliding Friction: Physical Principles and Applications”, Springer, Heidelberg, Germany.

Plimpton, S., 1995, “Fast parallel algorithms for short-range molecular dynamics”, Journal of Computational Physics v. 117, pp. 1-19.

Rapaport, D.C., 1995, “The Art of Molecular Dynamics Simulation”, Cambridge University Press, Cambridge, UK.

Singh, J.P., Holt, C., Totsuka, T., Gupta, A., Henessy, J., “Load-balancing and data-locality in adaptive hierarquical N-body methods: Barnes-Hut, fast-multipole, and radiosity”, Journal of Parallel and Distributed Computing v. 27, pp. 118-141.

Stephany, S., Correa, R.V., Mendes, C.L., Preto, A.J., 2000, “Identifying performance bottlenecks in a radiative transfer application, in Applications of High-Performance Computing in Engineering VI, eds. M. Ingber, H. Power, and C.A. Brebbia”, pp. 51-60, WIT Press, Southampton, UK.