Parallel Short Range Molecular Dynamics Simulations on Computer Clusters: Performance Evaluation and Modeling

T. E. KARAKASIDIS
Hydromechanics Laboratory, School of Engineering
University of Thessaly, Pedion Areos, 38334 Volos, Greece
thkarak@uth.gr

N. S. CHOLEVAS
Department of Mechanical and Industrial Engineering
School of Engineering, University of Thessaly
Pedion Areos, 38334 Volos, Greece

A. B. LIAKOPOULOS
Hydromechanics Laboratory, School of Engineering
University of Thessaly, Pedion Areos, 38334 Volos, Greece

Abstract—This paper describes the performance of a portable molecular dynamics code running on an eight-node PC cluster. The molecular dynamics code is based on the atom decomposition method for distributing the computation load among the processors and the MPI protocol for managing communications among processors. We discuss the changes made to the serial code with an effort to maintain its readability. We examined the program performance for system sizes of order $10^2$ to $10^4$ atoms and number of processors varying from 1 to 8, by measuring the total execution time and the corresponding speedup, as well as the communication time for data exchange and the time for the calculation of interatomic forces. Using simple communication and computation load considerations, we propose models in order to explain the observed behaviour and predict the optimal usage of the cluster. It turns out that using few parameters that can be easily measured one can predict quite accurately the optimal usage of small clusters running short range molecular dynamics programs. © 2005 Elsevier Ltd. All rights reserved.

Keywords—Molecular dynamics, Parallel simulations, Lennard-Jones liquid, Cluster computing, Distributed computing, MPI, Atom Decomposition.

1. INTRODUCTION

Molecular dynamics (MD) is a well-established simulation method based on an atomic description of matter. MD is well-suited for the study of transport and structural properties as it can probe microscopic mechanisms not easily accessible by experiment and thus, it is widely used in several areas of physics, materials science, engineering, and the study of biological systems. In the
framework of classical physics, MD provides us with a set of atomic trajectories (determined by the interatomic potential and Newton's laws of motion) and allows the calculation of system's properties through the formalism of statistical mechanics [1,2].

Even though MD is simple in its conceptual setting it is quite demanding in computer time if one attempts to simulate either very large systems or very long real times [3,4]. Hardware and algorithmic advances in parallel computing have provided the means to overcome these problems, to a certain extent. The introduction of parallel computers initiated a great deal of research towards the development of parallel codes for computationally intensive problems. However, the high cost of parallel computers had limited the availability of such machines to well-funded research centers. The emergence of Linux and the Beowulf cluster concept allowed the introduction of low-cost high-performance parallel computation to small research groups while many major labs created and operate vast such clusters. Clusters are easy to maintain and upgrade as they are built using top of the self components that are available by many providers. Another advantage of use of computer clusters is that their size can grow along with the computational needs and the available financial resources. On the other hand, one does not have total vendor support. However, there is often widely available relevant information sharing on the web. Typically, an inexpensive computer cluster consists of PCs running Linux or workstations running Unix. Node interconnection can be accomplished through a variety of networking such as Fast Ethernet to Gigabit Ethernet, Myrinet, or more sophisticated hardware [5]. Fast Ethernet is the simplest and less expensive solution, as Fast Ethernet switches have become very cheap.

Communication among processors takes place through a message passing system. A very often used system is the message passing interface MPI [6,7]. The user inserts calls to runtime optimized set of interprocessor communication routines. Independent copies of the identical compiled source are executed on processors of the cluster. This methodology assures portability of the parallelized programs on many high or low-end parallel systems as MPI has been established as a standard message passing system.

Several strategies to parallelize MD codes for a variety of potentials and systems have been given (see, for example, [3,8] and references there in). In moving from a sequential code to a parallel one, there are two major issues that one should pay attention: balancing of the computational load over the available processors and efficient communication of data among processors. In a short-range MD code, the former concerns mainly the calculation of forces and the construction of the list of neighbours while the latter concerns optimization of exchange of necessary data (positions and/or velocities). Very optimized codes that take advantage of a particular computer architecture may become unreadable for further development or lose portability.

Once the networked cluster in place and the code is parallelized, the next question is how many processors should one use in order to achieve optimal performance for the code. For a given size problem increasing the number of processors may not ameliorate the performance or may even make things worse.

There are communication models that try to explain and predict the performance of parallel machines that are connected through a network (see, for example, [9]). Extracting exact values for the model parameters is not an easy task and there is considerable work by computer specialists concerning measurement of relevant quantities (see, for example, [5,10-14]). In the majority of cases, communication measurements and modeling is the main purpose. It is true that the performance of an MD parallel program will be influenced by communication issues but computation and its relation with the communication part will also be very important issues too, as it is the case in every practical parallel program (see, for example, [15,16]). We address these issues having in mind the Molecular Dynamics practitioner and not the computer communication specialist.

In the present work, we deal with the parallelization of a sequential MD code (used for the simulation of simple liquids with short range interactions) using the MPI protocol on a small Beowulf PC cluster. We present the strategy used to parallelize the code with an effort to
Molecular Dynamics Simulations 785

maintain its readability for future development and we give some hints concerning timing of several parts of the code so that one can find useful rules of thumb before staring production runs. It is evident that communication among processors is an important issue that one should seriously take into account when designing a parallel version of a code. We also propose rules of thumb for the optimum utilization of a cluster for a given size problem. Finally, using simple models, we attempt an explanation of the observed behaviour, and predict the optimal use of given cluster for a given size problem.

The paper is organized as follows. In Section 2.1, we present the salient features of the molecular dynamics method along with the algorithm of atom decomposition used for the parallelization of the code. In Section 2.2, we present details concerning the physical system studied and in Section 2.3, the hardware and software configuration of our cluster is described. In Section 2.4, we present details concerning the timing procedure and cluster performance evaluation. Section 3 contains the obtained results along with the models proposed for modeling the performance of the cluster and for the estimation of its optimal usage. Finally, in Section 4, we present the corresponding conclusions.

2. PHYSICAL MODEL AND NUMERICAL IMPLEMENTATION

2.1. Molecular Dynamics and Atom Decomposition

In molecular dynamics, a system consists of N atoms which interact through Newton's second law and an appropriate interaction potential. If we limit our attention to cases of pair interactions, the equations of motion of the particles are given by

\[ F_i = m_i \frac{d^2 \mathbf{r}_i}{dt^2} = -\frac{\partial U(\mathbf{r}_1, \ldots, \mathbf{r}_N)}{\partial \mathbf{r}_i} = -\frac{\partial \sum_{ij} \phi(r_{ij})}{\partial \mathbf{r}_i} = \sum_j f_{ij}. \]  

The most time consuming part of an MD program is the calculation of the forces. Forces are obtained by calculating the interactions of a particle i with all particles j which lie within the cut-off range \( r_c \) of the potential. In order to accelerate this calculation, one does not check at each step if all atoms j lie within \( r_c \), but constructs for each particle i, the so-called neighbor list, a concept originally introduced by Verlet [17]. In this list, all atoms j that lie within a radius \( r_n \approx r_c + v \Delta t \) from a particle i are stored. Here, \( \Delta t \) is the timestep, \( n \) denotes the number of steps between list updates and \( v \) is the average speed of the particles at the given temperature. This way the interactions of particle i with particles j for the force calculation are limited for the next \( n \) steps to the atoms of this list.

Distribution of the computational load over the available processors can be accomplished by atom decomposition (AD), space decomposition, or force decomposition [3,18]. Given that AD seems to be quite efficient for small computer clusters [3] and in an attempt to maintain the readability of our code, we have chosen to use the AD method. In this approach, each of the \( P \) processors is assigned \( N/P \) particles to deal with which assures a good loadbalancing of the processors.

We parallelized the routine of the neighbor list and that of the force calculations. When calculating the forces in a serial code, we take advantage of Newton's third law, i.e., in the force routine, we calculate the interactions of particles i with all particles \( j > i \) and when we calculate the force \( F_i \) on particle i we add all the interactions \( f_{ij} \) to \( F_i \) and we also add \( -f_{ij} \) to \( F_j \), too. In the case of AD, as all neighbors are not necessarily on the same processor, we must communicate this information among processors. This would make loadbalancing more difficult to achieve as the processors dealing with the first particles would perform more calculations, and perhaps we would have an increased need for exchanging information between processors. It would also require considerable changes of the code [3]. We have concluded that it is not worth the complication for a small computer cluster.
The computation proceeds as following. At the first step, the positions of all particles are communicated between processors so each processor can calculate the forces due to the neighbors. Integration of equation (1) results in the new particle positions which then are communicated to all processors and the procedure is repeated.

2.2. Physical System and Computational Details

In this test case, we examined a physical problem already studied by other researchers as a benchmark case (see, for example, references [3,19,20]. We simulated a liquid for which interactions between atoms are described by a Lennard-Jones potential which is given by

$$\phi(r) = 4\varepsilon \left( \frac{\sigma}{r} \right)^{12} - \left( \frac{\sigma}{r} \right)^{6},$$

(2)

where $\varepsilon$ and $\sigma$ are constants, and $r$ denotes the interatomic distance. The simulations were performed in the microcanonical (NVE) ensemble. Forces were truncated at a distance $r_c = 2.5\sigma$ while the cutoff distance for the neighbor list was $r_n = 2.8\sigma$ and the update frequency ten steps. The positions and velocities of the particles were saved on disk every ten steps for post-processing reasons. The equations of motion were integrated using the Verlet algorithm which is accurate and stable with a timestep $10^{-14}$ s. The system was simulated at a reduced density $\rho^* = 0.8442$ and reduced temperature $T^* = 0.72$. Trajectories of particles and velocities where kept every five timesteps for post processing.

Simulations were performed using a parallelepiped box of size $L_x L_y L_z$ with periodic boundary conditions to simulate the bulk of a liquid. Atoms are put initially on an fcc lattice and are given velocities in order to achieve the desired temperature. The dimensions of the box are chosen so as to reproduce the desired density. The number of particles for the several systems studied varied from $N = 192$ up to $N = 10,800$.

The communication of the atom positions and velocities was performed in two ways. In the first one, we used the blocking Send and Receive commands of MPI [6,7]. In this case, the processor sends the desired information to the other processors and the program does not advance until the content is received by the destination processors. Particular attention is needed with the placement of these commands in the code as an unwise placement can lead to a deadlock, i.e., blockage of the program. In the second realization, we used the Broadcast command [6,7] which is a nonblocking command. In this case, the information is sent by one processor to all other processors.

2.3. Cluster Configuration

The Beowulf cluster consisted of eight PCs each equipped with an Athlon AMD processor at 1400 MHz with 512 KB L2 cache, 256 MB DDR RAM at 266 MHz, a 40 MB ULTRA ATA 100 hard disk with 2 MB cache, and a 100 Mbs Fast Ethernet switch. On the software side, the Suse 7.2 distribution of Linux, the MPICH v.1.2.4 of Argonne National Laboratory implementation of MPI and the GNU Fortran77 compiler version 2.95.3 were used.

2.4. Timing

Timing information was obtained via the MPI routine MPI_Wtime(). This routine returns a value that represents the number of seconds that have elapsed since some point in the past. It should be stressed that MPI_Wtime() returns wall-clock time [21], that is, it makes no allowance for such things as system time. So, if a process is interrupted by the system, the time it spends idle will be added into the elapsed time. The part of the program that we wanted to time was embedded between two calls to MPI_Wtime. The time of the procedure was calculated as the
Molecular Dynamics Simulations

The total execution time was measured as the maximum time over all processors in a given run. We measured also the average time of force calculation $T_{\text{force}}$ and the average time for data communication per cycle $T_{\text{comm}}$. The values reported here for $T_{\text{force}}$, and $T_{\text{comm}}$ are averages over 10000 integration steps. We also calculated the corresponding speedup which is the ratio of the total execution time for the parallel version over $P$ processors divided by the total execution time on a single processor. The sequential programs used here for timing purposes are identical to the parallel programs, except that they run on a single processor. During timing the MD application was the only user-launched process on the cluster apart system routines.

(a). Measured total execution time per timestep, $T_{\text{tot}}$, as a function of the number of processors ($P$) and system size ($N$) for Broadcast ($B$) and Send-Receive (SR) implementations.

(b). Zoom for small system size $N$.

Figure 1.
2.5
1.5
0.5

N=2304
N=4800
N=10800

(a). Measured total execution time per timestep, $T_{tot}$, as a function of the number of processors $P$ and system size $N$ using Broadcast methodology.

0.14 ....
0.12
0.1
0.08
0.06
0.04
0.02
0

1 2 4 6 8

(b). Zoom for small system size $N$.

Figure 2.

3. RESULTS AND DISCUSSION

Figure 1a shows the measured total execution time per timestep, $T_{tot}$, as a function of the number of processors ($P$) for several system sizes (i.e., number of atoms $N$). Measurements for both Broadcast and Send-Receive implementations are shown. We can see that the communication scheme affects the overall performance. In fact, the Broadcast methodology performs better than the Send-Receive methodology and this is clearly seen at the scale of the graph especially for $P = 8$. This is expected since the Send-Receive methodology involves more communication calls between processors.

In what follows, we focus on the results obtained using the Broadcast methodology. Focusing our attention to the systems with a large number of atoms we see that increasing $P$ results in a decrease of $T_{tot}$ (Figure 2a). However, in Figure 2b, we can see that for smaller system sizes $N$ the behaviour is rather complicated. For $N = 1176$ and $N = 864$, we have a reduction of the execution time for $P = 2$ but for larger values of $P$ we have an increase in the execution time. For the smaller systems studied, $N = 192$ and $N = 600$, the total execution time increases as a
function of the number of processors. This behavior is more clearly captured in a single graph Figure 3 where the corresponding speedups are presented for all system sizes studied as a function of the number of processors $P$. It is clear that for the $N = 192$, we have no speedup for $N = 600$, we have speedup for $P = 2$ but not for $P = 4, 6, 8$ and for $N = 864$, we have speedup for $P = 2, 4$ but not for $P = 6, 8$. For the larger systems we have speedup in all $P$ cases studied here but the speedup is not a monotonically increasing function of the number of processors, except for $N = 10,800$ and $N = 4,800$. This speedup behavior is typical of parallelized applications, as increasing the number of processors $P$ does not necessarily result in amelioration of the program performance.

To get a better insight of the inner workings of the algorithm and the computer cluster, we took a closer look at the parts where the program spends most of its execution time: the calculation of forces and the communication of new atom positions and velocities among processors.

In Figure 4, the force calculation time $T_{\text{force}}$ for each system size is plotted as a function of the number of processors, $P$. For fixed $N$, $T_{\text{force}}$ decreases as $1/P$. This behaviour is expected since in the atom decomposition method (see Section 2.1) the number of atoms that are assigned to each processor is $(N/P)$ and the burden of the calculation of interatomic forces is equally distributed among the processors.
0.2
0.15
0.1
0.05
0
0.2 4 6 8 10
P

Figure 5. Measured communication time per timestep, $T_{\text{comm}}$, as a function of number of processors $P$ and system size $N$ (Broadcast implementation).

Figure 6. Measured ratio $T_{\text{comm}}/T_{\text{force}}$ as a function of number of processors $P$ and system size $N$ for Broadcast methodology.

Figure 5 shows the corresponding results concerning communication time. The total communication time per timestep that is needed for the exchange of new atom positions and velocities among processors is plotted versus $P$. We can see that for all system sizes $N$ the communication time increases as the number of processors increases. This behaviour is expected since although increasing the number of processors results in smaller data packages to be transferred, more communications are needed in order that all data from one processor reach all others and each processor receives data from all others over the network which has a limited bandwidth. We can also see that for a fixed number of processors the communication time increases with the system size since the data packages to be transferred get bigger.

Figure 6 depicts the measured $(T_{\text{comm}}/T_{\text{force}})$ ratio. The ratio is larger than unity and it increases as $P$ increases, in accordance with the previously discussed behavior of $T_{\text{comm}}$ and $T_{\text{force}}$. However, we cannot get a clear idea about the relation of the $T_{\text{comm}}/T_{\text{force}}$ ratio with the speedups since cases where we have observed distinct speedup behavior exhibit the same $T_{\text{comm}}/T_{\text{force}}$ ratios (see for example, cases $N=600, 2304, 4800$).

In what follows, we present first a baseline model for the total execution time of the parallelized program based on simplifying assumptions on rates of communication and computation, and then we proceed to an improved model, having always in mind to balance accuracy of prediction with
ease of application by an MD practitioner. With these models we try not only to explain/predict the observed performance, but more importantly, from a practical point of view, to develop a tool for the optimal usage of a typical small cluster. Since, as we have seen, the Broadcast methodology performs better, the communication models described below deal only with this data-exchange methodology.

3.1. Performance Evaluation—The Baseline Model

3.1.1. Communication considerations

The communication time necessary for a problem decomposed to P processors can be expressed in terms of the total number of messages \( N_{\text{mess}} \) and the length of data necessary to exchange among processors \( N_{\text{shared}} \) (see [15]). In our case, we have double precision, i.e., 64-bit numbers, so we can write,

\[
T_{\text{comm}} = T_{\text{mess}} N_{\text{mess}} + (N_{\text{shared}} \times 64)/B,
\]

where \( B \) is the bandwidth of the network and \( T_{\text{mess}} \) is the necessary startup time for the communications routine. In fact, when a data package has to be transmitted, we have a call to the corresponding MPI routine. The time necessary to start up this transfer is related to the network latency but some models include also the so-called send and receive overheads of the message. The sum of these times is sometimes called end-to-end latency [22]. Note that for small clusters, the communication overheads are nearly constant and here we incorporate them into \( T_{\text{mess}} \).

In the case of our implementation, we try to make a rough estimate for \( T_{\text{comm}} \) due to exchanges through Broadcasts. We transmit positions and velocities in the following manner. Positions and velocities are stored in arrays of size \( X(N,3), V(N,3) \). Each processor deals with \( N/P \) atoms and thus in each iteration cycle once the processor has calculated the new positions and velocities we transmit them in rows of length \( N/P \). So, we have a total of six calls for sending and \( 6(P-1) \) for receiving from other processors. So, the total number of calls in each time step to Broadcast routines is \( N_{\text{mess}} = 6P \).

As far as the actual transfer of data is concerned, the traffic for one processor involves the following. Each processor sends six times \( N/P \) data packages to \( (P-1) \) processors and receives six \( (N/P) \) packages from \( (P-1) \) processors resulting in a total of \( 12(P-1)(N/P) \) packages (of double precision 64-bit numbers). This expressed in bits is \( 64 \times 12(P-1)N/P \) and the corresponding average time for the total communication time becomes

\[
T_{\text{comm}} = 6PT_{\text{mess}} + \frac{64 \times 12 \times (P-1) \times N}{P} \]

It is quite complicated to obtain reliable data concerning \( T_{\text{mess}} \) as it is not evident how to measure latency and overhead [10,22]. We take as an estimate for \( T_{\text{mess}} \) the nominal network latency of fast ethernet \( T_L = 150 \mu s \), and for \( B \), the nominal network bandwidth 100 Mbs. For this reason, we refer to \( T_{\text{mess}} \) as \( T_L \) further in the text. Using these values in equation (4), we obtain the results listed in Part (a) of Table 1. Parts (b) and (c) of Table 1 list the measured values for \( T_{\text{comm}} \) as well as the relative error in the model predictions. Although the model is quite simple we have at most a 58% relative error (except for the case \( N = 192, P = 2 \)).

In the rest of the paper, we refer to the first term of equation (4) as \( T_{\text{latency}} \) and to the second as \( T_{\text{transfer}} \), i.e., \( T_{\text{latency}} = 6PT_L \), and \( T_{\text{transfer}} = (1/B) \times 64 \times 12(P-1)N/P \).

In Table 2, we present the baseline model predictions for the ratio \( T_{\text{latency}}/T_{\text{transfer}} \) using \( B = 100 \text{ Mbs} \) and \( T_L = 150 \mu s \). It is interesting to note that the cases that correspond to values greater than 1 (shaded region) are the cases where we do not have speedup or that the speedup decreases for further increase in \( P \). Even cases with ratios less than 1.0 and greater than \( \approx 0.5 \) (boxed cases) show limited speedup \( (P = 2, N = 600), (P = 4, N = 864), (P = 6, N = 1176) \).
Table 1. Total communication time per timestep (in secs) (a) predicted by the baseline model, (b) measured values, and (c) relative error, as a function of system size \( N \) and number of processors \( P \).

<table>
<thead>
<tr>
<th>( P )</th>
<th>( N = 192 )</th>
<th>( N = 600 )</th>
<th>( N = 864 )</th>
<th>( N = 1176 )</th>
<th>( N = 2304 )</th>
<th>( N = 4800 )</th>
<th>( N = 10800 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2.50E-03</td>
<td>4.00E-03</td>
<td>4.96E-03</td>
<td>6.11E-03</td>
<td>1.02E-02</td>
<td>1.94E-02</td>
<td>4.14E-02</td>
</tr>
<tr>
<td>4</td>
<td>4.65E-03</td>
<td>6.90E-03</td>
<td>8.35E-03</td>
<td>1.01E-02</td>
<td>1.63E-02</td>
<td>3.00E-02</td>
<td>6.29E-02</td>
</tr>
<tr>
<td>6</td>
<td>6.57E-03</td>
<td>9.06E-03</td>
<td>1.07E-02</td>
<td>1.26E-02</td>
<td>1.95E-02</td>
<td>3.47E-02</td>
<td>7.13E-02</td>
</tr>
<tr>
<td>8</td>
<td>8.43E-03</td>
<td>1.10E-02</td>
<td>1.27E-02</td>
<td>1.47E-02</td>
<td>2.20E-02</td>
<td>3.80E-02</td>
<td>7.64E-02</td>
</tr>
</tbody>
</table>

(b) Measured

<table>
<thead>
<tr>
<th>( P )</th>
<th>( N = 192 )</th>
<th>( N = 600 )</th>
<th>( N = 864 )</th>
<th>( N = 1176 )</th>
<th>( N = 2304 )</th>
<th>( N = 4800 )</th>
<th>( N = 10800 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1.42E-03</td>
<td>3.43E-03</td>
<td>4.39E-03</td>
<td>5.76E-03</td>
<td>1.06E-02</td>
<td>2.21E-02</td>
<td>4.63E-02</td>
</tr>
<tr>
<td>4</td>
<td>3.02E-03</td>
<td>7.75E-03</td>
<td>1.06E-02</td>
<td>1.38E-02</td>
<td>2.55E-02</td>
<td>4.99E-02</td>
<td>1.07E-01</td>
</tr>
<tr>
<td>6</td>
<td>5.21E-03</td>
<td>1.23E-02</td>
<td>1.67E-02</td>
<td>2.32E-02</td>
<td>4.20E-02</td>
<td>7.97E-02</td>
<td>1.71E-01</td>
</tr>
<tr>
<td>8</td>
<td>6.32E-03</td>
<td>1.41E-02</td>
<td>1.87E-02</td>
<td>2.56E-02</td>
<td>4.65E-02</td>
<td>8.73E-02</td>
<td>1.80E-01</td>
</tr>
</tbody>
</table>

(c) Relative error (measured-model)/measured

<table>
<thead>
<tr>
<th>( P )</th>
<th>( N = 192 )</th>
<th>( N = 600 )</th>
<th>( N = 864 )</th>
<th>( N = 1176 )</th>
<th>( N = 2304 )</th>
<th>( N = 4800 )</th>
<th>( N = 10800 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>-75.94%</td>
<td>-16.52%</td>
<td>-13.15%</td>
<td>-6.04%</td>
<td>2.98%</td>
<td>12.22%</td>
<td>10.63%</td>
</tr>
<tr>
<td>4</td>
<td>-53.92%</td>
<td>11.01%</td>
<td>21.58%</td>
<td>26.89%</td>
<td>36.37%</td>
<td>39.93%</td>
<td>41.17%</td>
</tr>
<tr>
<td>6</td>
<td>-26.08%</td>
<td>26.48%</td>
<td>36.03%</td>
<td>45.85%</td>
<td>53.66%</td>
<td>56.44%</td>
<td>58.19%</td>
</tr>
<tr>
<td>8</td>
<td>-33.35%</td>
<td>21.76%</td>
<td>32.06%</td>
<td>42.47%</td>
<td>52.71%</td>
<td>56.53%</td>
<td>57.52%</td>
</tr>
</tbody>
</table>

Table 2. \( T_{\text{latency}}/T_{\text{transfer}} \) as a function of system size \( N \) and number of processors \( P \): baseline model predictions using \( B = 100 \text{ Mbs} \) and \( T_L = 150 \mu \text{s} \).

<table>
<thead>
<tr>
<th>( P )</th>
<th>( N=192 )</th>
<th>( N=600 )</th>
<th>( N=864 )</th>
<th>( N=1176 )</th>
<th>( N=2304 )</th>
<th>( N=4800 )</th>
<th>( N=10800 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2.56</td>
<td>0.82</td>
<td>0.57</td>
<td>0.42</td>
<td>0.21</td>
<td>0.10</td>
<td>0.05</td>
</tr>
<tr>
<td>4</td>
<td>3.41</td>
<td>1.09</td>
<td>0.76</td>
<td>0.56</td>
<td>0.28</td>
<td>0.14</td>
<td>0.06</td>
</tr>
<tr>
<td>6</td>
<td>4.61</td>
<td>1.47</td>
<td>1.02</td>
<td>0.75</td>
<td>0.38</td>
<td>0.18</td>
<td>0.08</td>
</tr>
<tr>
<td>8</td>
<td>5.85</td>
<td>1.87</td>
<td>1.30</td>
<td>0.96</td>
<td>0.49</td>
<td>0.23</td>
<td>0.10</td>
</tr>
</tbody>
</table>

This is an indication that latency related time plays an important role in the performance of the parallelized program. Very good speedups correspond to ratios less than 0.1.

Using the communications model, equation (4), we can estimate when the latency contribution becomes nearly equal to that of transfer. These estimates depend on the number of processors and are presented in Table 3.

In Table 4, all \( N/P \) sizes that we have studied in the present work are listed. The shaded region corresponds to the cases that we do not have speedup or speedup does not increase for further increase in \( P \) or even gets worse as \( P \) increases. Note that the values of \( N/P \) in the shaded region are lower than or equal to the \( (N/P) \) limiting values predicted by the baseline model in Table 3. The numbers in boxes correspond to case of limited speedup.

<table>
<thead>
<tr>
<th>( P )</th>
<th>( N/P ) limiting</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>246</td>
</tr>
<tr>
<td>4</td>
<td>164</td>
</tr>
<tr>
<td>6</td>
<td>147</td>
</tr>
<tr>
<td>8</td>
<td>140</td>
</tr>
</tbody>
</table>
Table 4. \( N/P \) sizes for various system sizes \( N \) as a function of number of processors \( P \).

<table>
<thead>
<tr>
<th>( P )</th>
<th>( N = 192 )</th>
<th>( N = 600 )</th>
<th>( N = 864 )</th>
<th>( N = 1176 )</th>
<th>( N = 2304 )</th>
<th>( N = 4800 )</th>
<th>( N = 10800 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>96</td>
<td>300</td>
<td>432</td>
<td>588</td>
<td>1152</td>
<td>2400</td>
<td>5400</td>
</tr>
<tr>
<td>4</td>
<td>38</td>
<td>150</td>
<td>216</td>
<td>294</td>
<td>576</td>
<td>1200</td>
<td>2700</td>
</tr>
<tr>
<td>6</td>
<td>32</td>
<td>100</td>
<td>144</td>
<td>196</td>
<td>384</td>
<td>800</td>
<td>1800</td>
</tr>
<tr>
<td>8</td>
<td>24</td>
<td>75</td>
<td>108</td>
<td>147</td>
<td>288</td>
<td>600</td>
<td>1350</td>
</tr>
</tbody>
</table>

Table 5. \( T_{\text{force(model)}} / T_{\text{latency(model)}} \) as a function of system size \( N \) and number of processors \( P \).

<table>
<thead>
<tr>
<th>( P )</th>
<th>( N = 192 )</th>
<th>( N = 600 )</th>
<th>( N = 864 )</th>
<th>( N = 1176 )</th>
<th>( N = 2304 )</th>
<th>( N = 4800 )</th>
<th>( N = 10800 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.42</td>
<td>0.87</td>
<td>1.42</td>
<td>2.15</td>
<td>3.19</td>
<td>6.45</td>
<td>26.17</td>
</tr>
<tr>
<td>4</td>
<td>0.12</td>
<td>0.24</td>
<td>0.39</td>
<td>0.59</td>
<td>0.85</td>
<td>1.65</td>
<td>6.54</td>
</tr>
<tr>
<td>6</td>
<td>0.06</td>
<td>0.12</td>
<td>0.18</td>
<td>0.27</td>
<td>0.39</td>
<td>0.76</td>
<td>2.91</td>
</tr>
<tr>
<td>8</td>
<td>0.03</td>
<td>0.07</td>
<td>0.11</td>
<td>0.17</td>
<td>0.23</td>
<td>0.43</td>
<td>1.63</td>
</tr>
</tbody>
</table>

In Table 5, we present data on the ratio \( T_{\text{force(model)}} / T_{\text{latency(model)}} \). Note that for very small values of the ratio we do not have speedup while for large values we have significant speedup. For a given problem size, this ratio decreases as \( P \) increases and when it becomes lower than 1.0 we do not have any speedup amelioration.

This behaviour could be explained qualitatively by noting that as the number of processors increases the computational load per processor decreases while the latency time increases since the number of required communication routine calls increases. Consequently, beyond some point each processor spends time idle for the data transfer to be completed in order to resume calculations.

### 3.1.2 Computation time considerations

The main computational load in a Molecular Dynamics simulation program comes from calculation of the interatomic forces and the construction of the neighbor list. The force calculation scales linearly with \( N \) for short range interactions as every atom has a limited number of neighbours, \( N_c \), within the cut-off radius \( r_c \). \( N_c \) is fixed for a given density, so the time spent for force calculation per processor is

\[
T_{\text{force}} = \frac{N}{P} N_c t_F, \quad (5)
\]

where \( (N/P) \) is the number of atoms attributed to each of the \( P \) processors and \( t_F \) is the time necessary to calculate the interaction force between two atoms.

The calculation of neighbours on the other hand in our case is done with a Verlet list so for every atom assigned to the processor a check of the distance of all other atoms but itself is performed, every \( N_{\text{list}} \) steps. So, the average time spent per processor will be

\[
T_{\text{neighb}} = \frac{1}{N_{\text{list}}} N (N-1) t_N, \quad (6)
\]

where \( t_N \) is the time necessary to calculate the distance between two atoms.

To a good degree of approximation, the total computation time per timestep is the sum of the time necessary for the force and neighbor calculations per timestep, i.e.,

\[
T_{\text{comp}} = T_{\text{force}} + T_{\text{neighb}}. \quad (7)
\]

Good estimates of \( t_F \) and \( t_N \) can be determined by direct timing. In our case, \( N_c = 60 \), \( t_F = 3 \times 10^7 \text{s} \), and \( t_N = 1.2 \times 10^{-7} \text{s} \).

Taking into account the communication model that has been already discussed, i.e.,

\[
T_{\text{comm}} = T_{\text{latency}} + T_{\text{transf}} = 6P T_L + \frac{64 \times 12 \times (P-1) N}{B P}, \quad (8)
\]
Table 6. Total execution time per timestep $T_{tot}$ (in secs) estimated by the baseline model as a function of the system size $N$ and number of processors $P$ and the corresponding percent difference with reference to measured values.

<table>
<thead>
<tr>
<th>$P$</th>
<th>$N = 192$</th>
<th>$N = 600$</th>
<th>$N = 864$</th>
<th>$N = 1176$</th>
<th>$N = 2304$</th>
<th>$N = 4800$</th>
<th>$N = 10800$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4.45E-03</td>
<td>1.16E-02</td>
<td>1.72E-02</td>
<td>2.50E-02</td>
<td>6.28E-02</td>
<td>2.01E-01</td>
<td>8.38E-01</td>
</tr>
<tr>
<td>4</td>
<td>5.63E-03</td>
<td>1.07E-02</td>
<td>1.45E-02</td>
<td>1.95E-02</td>
<td>4.25E-02</td>
<td>1.21E-01</td>
<td>4.61E-01</td>
</tr>
<tr>
<td>6</td>
<td>7.22E-03</td>
<td>1.16E-02</td>
<td>1.48E-02</td>
<td>1.89E-02</td>
<td>3.70E-02</td>
<td>9.52E-02</td>
<td>3.37E-01</td>
</tr>
<tr>
<td>8</td>
<td>8.92E-03</td>
<td>1.29E-02</td>
<td>1.58E-02</td>
<td>1.95E-02</td>
<td>3.51E-02</td>
<td>8.33E-02</td>
<td>2.76E-01</td>
</tr>
</tbody>
</table>

Percent Difference From Measured Speedup

<table>
<thead>
<tr>
<th>$P$</th>
<th>$N = 192$</th>
<th>$N = 600$</th>
<th>$N = 864$</th>
<th>$N = 1176$</th>
<th>$N = 2304$</th>
<th>$N = 4800$</th>
<th>$N = 10800$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>-55.41%</td>
<td>-21.70%</td>
<td>-10.65%</td>
<td>-0.90%</td>
<td>10.35%</td>
<td>20.16%</td>
<td>25.28%</td>
</tr>
<tr>
<td>4</td>
<td>-42.69%</td>
<td>5.63%</td>
<td>14.87%</td>
<td>19.56%</td>
<td>25.15%</td>
<td>27.94%</td>
<td>30.06%</td>
</tr>
<tr>
<td>6</td>
<td>-20.96%</td>
<td>22.89%</td>
<td>30.36%</td>
<td>38.26%</td>
<td>41.83%</td>
<td>40.23%</td>
<td>37.81%</td>
</tr>
<tr>
<td>8</td>
<td>-27.51%</td>
<td>20.98%</td>
<td>29.39%</td>
<td>38.21%</td>
<td>44.29%</td>
<td>43.54%</td>
<td>39.77%</td>
</tr>
</tbody>
</table>

Table 7. Baseline model estimated speedups and percent difference from measured speedups.

<table>
<thead>
<tr>
<th>$P$</th>
<th>$N = 192$</th>
<th>$N = 600$</th>
<th>$N = 864$</th>
<th>$N = 1176$</th>
<th>$N = 2304$</th>
<th>$N = 4800$</th>
<th>$N = 10800$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.88</td>
<td>1.31</td>
<td>1.42</td>
<td>1.51</td>
<td>1.67</td>
<td>1.81</td>
<td>1.90</td>
</tr>
<tr>
<td>4</td>
<td>0.69</td>
<td>1.42</td>
<td>1.69</td>
<td>1.94</td>
<td>2.47</td>
<td>3.01</td>
<td>3.45</td>
</tr>
<tr>
<td>6</td>
<td>0.54</td>
<td>1.30</td>
<td>1.66</td>
<td>2.00</td>
<td>2.84</td>
<td>3.81</td>
<td>4.73</td>
</tr>
<tr>
<td>8</td>
<td>0.44</td>
<td>1.17</td>
<td>1.55</td>
<td>1.94</td>
<td>2.99</td>
<td>4.35</td>
<td>5.78</td>
</tr>
</tbody>
</table>

Percent Difference From Measured Speedup

<table>
<thead>
<tr>
<th>$P$</th>
<th>$N = 192$</th>
<th>$N = 600$</th>
<th>$N = 864$</th>
<th>$N = 1176$</th>
<th>$N = 2304$</th>
<th>$N = 4800$</th>
<th>$N = 10800$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>-9.71%</td>
<td>-13.15%</td>
<td>-4.53%</td>
<td>-2.58%</td>
<td>-0.39%</td>
<td>-0.50%</td>
<td>4.92%</td>
</tr>
<tr>
<td>4</td>
<td>-19.50%</td>
<td>-45.92%</td>
<td>-35.86%</td>
<td>-28.66%</td>
<td>-20.23%</td>
<td>-11.35%</td>
<td>-1.58%</td>
</tr>
<tr>
<td>6</td>
<td>-40.96%</td>
<td>-78.58%</td>
<td>-66.08%</td>
<td>-67.65%</td>
<td>-54.71%</td>
<td>-34.24%</td>
<td>-14.24%</td>
</tr>
<tr>
<td>8</td>
<td>-53.72%</td>
<td>-74.27%</td>
<td>-63.82%</td>
<td>-67.50%</td>
<td>-61.53%</td>
<td>-42.11%</td>
<td>-17.96%</td>
</tr>
</tbody>
</table>

The total execution time per timestep can be estimated using

$$T_{tot} = T_{comp} + T_{comm},$$

and is listed in Table 6.

The corresponding estimated speedups appear in Table 7. It is evident that the baseline model overestimates the speedups.

Using the model for the total execution time we can also estimate the optimum number of processors to be used for a given $N$, or equivalently the optimum $N/P$, above which we do not obtain any amelioration of the speedup. Setting

$$\frac{dT_{tot}}{dP} = 0,$$

we obtain

$$N \left[ \frac{N}{P} \right]_{opt} = \sqrt{\frac{N_t F + (1/N_{tot}) (N - 1) t_N - (12 \cdot 64) / B}.}

The calculated values of $(N/P)_{opt}$ and $P_{opt}$ for the various systems sizes $N$ appear in Table 8.

For the cases with $N \leq 864$, the baseline model predictions agree reasonably well with the measured data. However, for cases with $N \geq 1176$, the predictions seem to deviate considerably from the measurements.
Table 8. Baseline model predictions for the optimum number of processors for a given size problem $N$ and the corresponding optimum $(N/P)$.

<table>
<thead>
<tr>
<th>$P$</th>
<th>$N = 192$</th>
<th>$N = 600$</th>
<th>$N = 864$</th>
<th>$N = 1176$</th>
<th>$N = 2304$</th>
<th>$N = 4800$</th>
<th>$N = 10800$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_{\text{opt}}$</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>10</td>
<td>19</td>
<td>41</td>
</tr>
<tr>
<td>$(N/P)_{\text{opt}}$</td>
<td>115</td>
<td>174</td>
<td>192</td>
<td>207</td>
<td>233</td>
<td>252</td>
<td>263</td>
</tr>
</tbody>
</table>

Assuming that the bandwidth is constant, equation (11) can be further simplified by noting that the relative contribution of the term $(12 \cdot 64/B)$ becomes small as the system size $N$ increases. If we neglect this term, equation (11) becomes

$$
\left( \frac{N}{P} \right)_{\text{opt}} = \sqrt{\frac{6Nt_L}{N_C t_F + (1/N_{\text{list}})(N-1)t_N}}.
$$

and we see that $(N/P)_{\text{opt}}$ and $P_{\text{opt}}$ depend on the ratio of the latency to the force calculation time (since $t_N$ can be related to $t_F$ by a simple operation count). This means that the network latency and the processor speed are the two most important parameters that one should take into account in order to achieve optimal performance on a cluster. This is in qualitative agreement with the observations presented in Section 3.1 where we have seen that the ratio $T_{\text{latency}}/T_{\text{force}}$ played an important role in the performance of the parallel program.

### 3.2. Improving the Performance Model

In this section, we attempt to improve the performance model described in Section 3.1 by allowing $T_{\text{latency}}$ and bandwidth $B$ to become functions of $P$. There is experimental evidence that this is the case, see, for example, reference [5] and the discussion below. In Figure 7, the measured total communication time $T_{\text{comm}}$ is plotted versus the data package length $64N/P$. We can see that for each case $P = 2, 4, 6, 8$ the communication time $T_{\text{comm}}$ seems to follow a linear behaviour but with a different slope in each case. Thus, it seems reasonable to assume that

$$
T_{\text{comm}} = T_{\text{latency}}(P) + \frac{N \cdot 64}{P} \frac{1}{B(P)}
$$

and obtain $T_{\text{latency}}(P)$ and $B(P)$ by the method of least-squares.

For the latency (in seconds), we obtain

$$
T_{\text{latency}}(P) = 0.0008P - 0.0009
$$
Table 9. (a) Total execution time per timestep $T_{tot}$ (in secs) estimated by the improved model as a function of the system size $N$ and the number of processors $P$ (b) corresponding percent difference with reference to measured values.

<table>
<thead>
<tr>
<th>$P$</th>
<th>$N = 192$</th>
<th>$N = 600$</th>
<th>$N = 864$</th>
<th>$N = 1176$</th>
<th>$N = 2304$</th>
<th>$N = 4800$</th>
<th>$N = 10800$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3.48E-03</td>
<td>1.09E-02</td>
<td>1.67E-02</td>
<td>2.47E-02</td>
<td>6.33E-02</td>
<td>2.03E-01</td>
<td>8.45E-01</td>
</tr>
<tr>
<td>4</td>
<td>4.97E-03</td>
<td>1.14E-02</td>
<td>1.60E-02</td>
<td>2.21E-02</td>
<td>4.89E-02</td>
<td>1.35E-01</td>
<td>4.96E-01</td>
</tr>
<tr>
<td>6</td>
<td>7.11E-03</td>
<td>1.44E-02</td>
<td>1.95E-02</td>
<td>2.59E-02</td>
<td>5.22E-02</td>
<td>1.28E-01</td>
<td>4.14E-01</td>
</tr>
<tr>
<td>8</td>
<td>9.43E-03</td>
<td>1.81E-02</td>
<td>2.40E-02</td>
<td>3.13E-02</td>
<td>5.99E-02</td>
<td>1.37E-01</td>
<td>3.98E-01</td>
</tr>
</tbody>
</table>

(b) Percent difference from measured values

<table>
<thead>
<tr>
<th>$P$</th>
<th>$N = 192$</th>
<th>$N = 600$</th>
<th>$N = 864$</th>
<th>$N = 1176$</th>
<th>$N = 2304$</th>
<th>$N = 4800$</th>
<th>$N = 10800$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>-21.53%</td>
<td>-14.37%</td>
<td>-7.33%</td>
<td>0.34%</td>
<td>9.71%</td>
<td>19.31%</td>
<td>24.73%</td>
</tr>
<tr>
<td>4</td>
<td>-25.90%</td>
<td>-0.50%</td>
<td>5.63%</td>
<td>8.80%</td>
<td>13.96%</td>
<td>19.19%</td>
<td>24.81%</td>
</tr>
<tr>
<td>6</td>
<td>-19.14%</td>
<td>3.92%</td>
<td>7.89%</td>
<td>15.29%</td>
<td>17.93%</td>
<td>19.32%</td>
<td>23.64%</td>
</tr>
<tr>
<td>8</td>
<td>-34.81%</td>
<td>-10.83%</td>
<td>-7.46%</td>
<td>0.61%</td>
<td>4.89%</td>
<td>7.24%</td>
<td>12.97%</td>
</tr>
</tbody>
</table>

Table 10. Prediction of optimum $P$ and $N/P$ based on the improved model (equation (19)).

<table>
<thead>
<tr>
<th>$P$</th>
<th>$N = 192$</th>
<th>$N = 600$</th>
<th>$N = 864$</th>
<th>$N = 1176$</th>
<th>$N = 2304$</th>
<th>$N = 4800$</th>
<th>$N = 10800$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_{opt}$</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>8</td>
</tr>
</tbody>
</table>
| $(N/P)_{opt}$ | 107       | 223       | 284       | 348        | 537        | 837        | 1321        

and for the bandwidth (in Mbits per second), we obtain the power-law approximation

$$ B(P) = 3 \times 10^7 P^{-2.0285}. \quad (15) $$

Note that in equation (14) a linear functional relation with the number of processors reappears for $T_{latency}$ as we had supposed in the baseline model of Section 3.1 (see equation (4)). Furthermore the leading term coefficient, $8 \times 10^4 s$ in equation (14), is close to the coefficient in the baseline model $(6 \times 1.5 \times 10^{-4} s = 9 \times 10^{-4} s)$.

Substituting equations (14) and (15) into equation (13) and equations (5) and (6) into equations (7) and (9), we form a new model for the estimation of the total execution time $T_{tot}$. The new estimates of $T_{tot}$ and the associated relative errors are listed in Table 9. Comparing with the relative errors in Table 6 (for the baseline model) we can see that the results are significantly improved.

Without significant loss of accuracy, we can simplify equations (14) and (15) and write

$$ T_{latency}(P) = 0.0008 P \quad (16) $$

and

$$ B(P) = 3 \times 10^7 P^{-2.0}. \quad (17) $$

Substituting in equation (13), we obtain

$$ T_{comm} = 0.0008 P + \frac{N \times 64}{P} \frac{1}{3 \times 10^7 P^{-2.0}} = 10^{-4} \left[ 8 + \frac{64}{30} 10^{-2} N \right] P \quad (18) $$

Using the $T_{comp}$ of the previous model but the new $T_{comm}$ we can obtain a new estimate of the optimum values for $P$ and $(N/P)$. Differentiating the corresponding $T_{tot}$ with respect to $P$ and setting the derivative equal to zero we obtain for $N/P$

$$ \left( \frac{N}{P} \right)_{opt} = \sqrt{10^{-4} \left( \frac{8}{N} + \frac{64}{32} \right) \frac{10^{-2}}{N \cdot t_F + (1/N_{list}) (N-1) t_N} \quad (19) $$

The corresponding values for $P_{opt}$ and $(N/P)_{opt}$ appear in Table 10.
The predictions of the optimum number of processors are significantly improved and they approach our observed values, while the optimal computational load per processor \((N/P)_{opt}\) is also correctly predicted. It is of interest to note that in equation (19), \((N/P)_{opt}\) depends on the ratio of a "latency-like" parameter to \(t_F\) (or equivalently \(t_N\) since \(t_F = \text{const} \times t_N\)) as in the baseline model.

Returning to a qualitative discussion of the cluster performance we have seen that when the size of the physical system \(N\) is small, the calculations of new particle positions are performed quickly on each processor and the communication of the information among processors is done quite frequently penalizing the performance of the program due to the latency time which is comparable to the calculation time. As the size of the system increases, there is more time spent by each processor on force calculations and as a result the code spends more time on calculations than on calls for data transfers and thus it becomes more efficient. Increasing too much the number of processors for a given physical system size does not ameliorate performance by an important factor or can even make things worse since the computational load on each processor decreases but the communications with the increasing number of processors for the data exchange increase, penalizing the program performance. For each system size \(N\), there is an optimal number of processors, \(P_{opt}\), for the simulation or equivalently an optimal distribution of atoms over the processors \((N/P)_{opt}\). Replacing old processors by faster processors (i.e., reducing \(t_F\) and \(t_N\)) in a cluster will result in larger \((N/P)_{opt}\), i.e., the optimum usage for a given size physical system \(N\) will occur at a lower number of processors \(P\), if one maintains the same hardware for network connections.

Using our approach, if one knows \(N_c\), \(t_N\), \(t_F\), which can be obtained from measurements on the execution of the sequential code, and also measures \(T_{comm}\) (or \(T_{com}\) and \(B\)) as a function of the number of processors it is possible to construct an empirical model that can predict quite accurately the scaling behaviour of the execution time as function of \(P\) and \(N\), at least for small clusters.

4. CONCLUSIONS

We parallelized a Molecular Dynamics code for short-range molecular dynamics simulations using the atom decomposition method, with a view to maintaining the code structure relatively simple. We run the code on a small Beowulf cluster and examined the code speed-up for a number of particles ranging from \(N = 192\) to \(10,800\) and for number of processors from \(P = 2\) up to 8. As expected, the "Broadcast" communication methodology performs better than the Send and Receive implementation especially as the number of processors increases. A crucial parameter is found to be the latency of network communications compared to the time of force calculation. Based on simple computational and communication considerations, we constructed two models in order to explain the observed behaviour and predict the optimal usage of processors. In the first model, the communication part is based on assumptions about network latency and bandwidth while in the second the communication parameters are extracted from measurements. In both models, the computation time estimation uses as parameters the required times for force and neighbor calculation. The first model seems to explain the observed performance but overestimates the speedups. The second one predicts more accurately the optimal computational load per processor. Both models predict that the ratio of the force-calculation time to that of network latency determines the cluster performance. We believe that the described cluster performance modeling methodology can be used in any parallel application where the computational load can be divided equally among processors.

REFERENCES