A Parallel Engine for Graphical Interactive Molecular Dynamics Simulations

Eduardo Rocha Rodrigues
Post-graduation Program in Applied Computing
Brazilian Institute for Space Research (INPE)
 erocha@lac.inpe.br

Airam Jonatas Preto, Stephan Stephany
Lab for Computing and Applied Math (LAC)
Brazilian Institute for Space Research (INPE)
 [airam,stephan]@lac.inpe.br

Abstract

The current work proposes a parallel implementation for interactive molecular dynamics simulations (MD). The interactive capability is modeled by finite automata that are executed in the processing nodes. Any interaction implies in a communication between the user interface and the finite automata. The ADKS, an interactive sequential MD code that provides graphical output was chosen as a case study. A parallel version of this code was developed using the MPI communication library to check its parallel performance without/with visualization. Performance results are discussed for both cases and the influence of visualization in the performance is also treated, including image update rate. In order to allow a modular approach, a new parallel version of the ADKS is being implemented employing the PyMPI Python extension.

1. Introduction

Molecular Dynamics (MD) refer to computational techniques for simulations of particles as atoms and molecules. Resultant forces are calculated for each particle in order to estimate its path according to classical mechanics laws. Macroscopic system properties can thus be inferred [5]. Semi-classical corrections can be included to treat quantum effects [1]. This kind of simulation is similar to the study of celestial bodies behaviour in Astrophysics, commonly referred as N-body problem. However, MD normally deal with forces originated by short range potentials, while Astrophysics deal with a long range potential, the gravity. Resolution methods were optimized differently for long and short range potentials. Many areas employ MD, as in nanotechnology [4], biochemistry [9], molecular biology [10], and materials science [6].

MD simulations are computationally heavy in two aspects - the number \( N \) of particles that model the system and the number of timesteps. In order to reproduce actual microscopic behavior, timesteps are limited to the femtosecond scale and many thousands or millions of particles are required to achieve simulations that model picoseconds of sub-micron scale phenomena [13]. Standard short-range MD algorithms like cell-subdivision and neighbour-list reduce the complexity of the problem [14], but normally parallel algorithms are required and most common paradigms are the distribution of atoms, cells or force calculations among processors [13]. In Astrophysics, the Barnes-Hut algorithm is usually employed [15].

MD simulations can be implemented as programs that are executed in batch mode. An input data, execution, results analysis, and input data feedback cycle represents this kind of simulations. An alternative is to have an interactive capability, normally supported by an execution-time visualization. Such an interactive simulation allow to modify simulation parameters and to observe the resulting effect. Complex phenomena that could not be observed in an experiment or in the pos-morten analysis of a batch simulation can thus be studied in this kind of computational environment [11]. The need of parallel computing is emphasized, in order to have interactive simulations with execution times that are comfortable for the user.

A question that naturally arises concerning an interactive simulation in a parallel environment is how to control the simulation engine that is executed on several processors in face of the user input. For example, the Falcon system [3] is based on a set of tools and libraries, and an user interface that provide means to gather processed data, analyse data and change simulation parameters in execution time. Another system is based on the SPApSM (Scaleable, Parallel, Short-Range Molecular Dynamics) code [2]. SPApSM creates an interactive environment employing the Python extensible scripting language that supports the inclusion of visualization and user-developed modules. A similar system is shown in [8].

The current work proposes a MPI-based parallel implementation of interactive MD. An interactive sequential MD code, the ADKS [11], that provides graphical output using the MOTIF library, was chosen as a case study. The interactive capability is modeled by finite automata that are ex-
executed in each processor. User interaction triggers transitions in the finite automata. The user interacts with the simulation at the master node. This interaction produces messages that are sent to the processing nodes. Such messages contain user modified simulation parameters. Concurrently, processing nodes must periodically send updated particle positions to the master node. Any interaction implies in a communication between the user interface and the finite automata being executed at the processing nodes.

A parallel version of the ADKS was developed using the MPI communication library and the MOTIF-based visualization. As the ADKS sequential code is based on cell subdivision of the spatial domain, this version was parallelized by distributing sets of cells among processors. Performance results are discussed for the ADKS parallel version without and with visualization. Visualization aspects concerning image update rate and its influence on overall performance are also treated.

In order to allow a modular approach, a new parallel version of the ADKS is being implemented employing the PyMPI [12] Python extension. It would automatically provide user interaction with the finite automata at the processing nodes. A PyMPI parallel version with no visualization is already implemented.

2. Molecular Dynamics

The interaction between each pair of particles is given by a strong short-distance repulsion due to the Pauli exclusion principle and by a weak attraction that starts from a certain distance due to the van der Waals potential. Each phenomenon is modeled by a particular potential, according to the effect under study. For instance, the Coulomb potential is employed for electrically charged particles or both short and long range potentials appear in ion simulations [7]. A possible approach for noble gases or generic substances is the Lennard-Jones potential [14]. This is a short-range potential, being defined in function of the distance \( r \) between particles and of two specific parameters: \( \sigma \) and \( \epsilon \). The first parameter defines a length scale, while the second governs the strength of the interactions.

\[
u(r) = 4\epsilon \left( \frac{\sigma}{r} \right)^{12} - \left( \frac{\sigma}{r} \right)^{6}\]

(1)

Considering this equation, it can be shown that the Lennard-Jones potential is negligible for \( r > 3\sigma \). Thus, a cut-off distance \( r_c \) can be defined in order to have \( u(r) = 0 \) for \( r > r_c \). The interaction force \( \mathbf{f} \) derives from the potential according to \( \mathbf{f} = -\nabla u(r) \). Therefore, the number of calculations can be drastically reduced as \( r_c \) is very small in comparison with the dimension of the system of particles. For each particle, the interaction forces with all surrounding particles has to be computed in order to obtain the resultant force. Newton’s 2nd law is then employed to calculate the acceleration and successive numerical integrations updates velocity and position using the finite-difference method. Two simple schemes can be employed for these updating, the Verlet or the Leap-frog with good accuracy in comparison to more sophisticated schemes [14]. The relationship between the microscopic information that results from MD simulations and the macroscopic properties of the system is made by means of Statistical Mechanics. Systems can be identical, with equal energy or temperature, but particle behaviour is surely different. Macroscopic properties come from time-independent statistical averages. The thermodynamical state can be described by a set of parameters as temperature, pressure and number \( N \) of particles. Thus, in order to describe the particles, \( 6N \) variables are required (3 coordinates and 3 velocity components). These variables are part of the multidimensional phase space of the system. Each \( 6N \) set represent an instance of the ensemble of the system.

The accuracy of the simulation is influenced by the number \( N \) of particles and the number of timesteps. A too small \( N \) lead to fluctuations in the results [14], while a too large \( N \) may preclude the simulation due to high processing times. The profiling of any MD code show that the calculation of interaction forces between particles dominates the computation, taking about 99% of total time. Furthermore, if no optimization is used and all interaction forces are calculated, the number of computations is \( N(N-1)/2 \), i.e. \( O(N^2) \). This is very inefficient for short range potentials as most of the computations are related to negligible forces.

In such cases, the performance is improved by a rather common algorithm, the cell subdivision [14]. The spatial domain, for instance a 2D \( L \times L \) space is divided into square cells (in a 3D domain, cubic cells) in a way that particles in a cell only interacts with particles of neighbouring cells, i.e. the side of the cell cannot be smaller than the cutoff distance \( r_c \). Normally, a side of \( L/r_c \) is taken. As shown in Fig. 1(a), for a short range potential, the resultant force for a given particle is only due to particles that are inside the depicted circle of radius \( r_c \).

Cell subdivision is shown in Fig. 1(b), where force calculation is performed sweeping cells row by row from left to right in a downward direction. This scheme is further improved by using Newton’s 3rd law to avoid redundant calculations. For instance, considering the central cell of the \( 3 \times 3 \) marked area with 9 cells, force calculation needs to be performed for only the 5 shaded cells in order to determine forces between particles inside the \( r_c \) circle. Forces for the particles of the remaining 4 (not-shaded) cells are already known due to the sweep of previous rows.

In this case, it can be shown that number of computa-
tions is $O(N)$ [16]. However, even considering cells, some calculations are wasted as some particles in adjacent cells are beyond $r_c$. This can be avoided by the use of a neighbour list for each particle. This list keeps track of all particles inside a $r_n = r_c + \Delta r$ radius. $\Delta r$ is small in comparison to $r_c$, but must be chosen accordingly to the neighbour list update rate, in order to take into account outside particles that may move into the $r_c$ range [14]. The scheme is depicted in Fig. 1(c), where the inner circle has radius $r_n$ and the outer, radius $r_n$. The neighbour list is implemented as a linked list and reduces the area to be analyzed by a ratio of $\pi/9$ (or $4\pi/81$ in 3D). This is simply the ratio between the marked square and its inscribed circle. The neighbour list can also be used in the all pairs algorithm.

Three parallel algorithms for short range MD are described in [13]. The first, atom decomposition, distributes $N$ particles among $p$ processors, each processor calculating forces and updating the positions of its set of particles. Neighbour list is employed to reduce the number of calculations. However, particles may interact with others in any processor and thus all processors need updated positions of all particles to calculate forces. The second algorithm divides forces among processors, each one computing forces between subsets of particles. Particle position update is only required for the calculation of local forces. Finally, the third algorithm, spatial decomposition, divides the domain in cells that are assigned to the processors. However, according to the simulation, particles may migrate in a way that some cells may contain more particles than others, causing load unbalancing.

3. The ADKS simulator

The ADKS simulator is an interactive program with visualization capability [11] developed for the study of fractures and grain boundary behaviour in solids, as shown in Fig. 2. It was developed by INPE in cooperation with the Helsinki Tech University. Particles are displayed and updated according to an user-defined number of timesteps. Simulation parameters can be modified during the simulation in order to allow the study of their influence, providing a steering capability for the simulation. Furthermore, this program allows in depth analysis of complex transient phenomena, such as material fractures.

The ADKS program consists essentially of three modules: an MD simulation algorithm, a graphics window for output of data and an user interface for input of parameters. These modules are integrated in an event-driven environment which handles the exchange of information between modules, and allow the processing of any actions requested by the user during the simulation. The program was written in C and developed for an X11 Window System platform. All graphics are based on the MOTIF library. A simu-

![Figure 1. MD methods for short range interaction force calculation.](image)
lation may be paused by user-triggered events and resumed thereafter.

An immediate advantage of the ADKS strategy is to save storage. If simulation data were dumped to a disk, the coordinates of all particles for all timesteps would require hundreds or even thousands of megabytes for a typical MD simulation. The addition of velocity data would double this requirement. In the interactive approach, otherwise, the continuous updating of data to the screen combined with the possibility of storing configurations in a selective way keep the required disk space at a minimum. Another ADKS feature is the way which equilibrium or near-equilibrium states may be identified by direct visual inspection. This is desired in many circumstances, for instance when studying the effect of new conditions, such as variations in the stress intensity or temperature, on the equilibrium state. Similar studies in a conventional simulation approach would require a large number of iterations of the run-simulation/analyse-data/run-simulation cycle. However, the best advantage of ADKS is obtained in the analysis of systems that present dynamic effects, such as in the case of material fractures. Post-processing analysis of data would be very limited and a full animation would present storage problems without the benefits of interactivity. For example, the ADKS has the ability to follow the development of special phenomena such as the emission and propagation of dislocations in a lattice of a polycrystalline material.

4. Interactive MD simulations in a parallel environment

In an interactive simulation, executed on a sequential machine, a simple event-driven programming model allows steering, i.e. the modification of simulation parameters without restarting it from the beginning. A parallel version of this model for shared memory machines would simply require to decouple the visualization interface from the simulation engine. Similarly to the sequential model, events can modify parameters of the simulation that is executed by multiple threads, provided that race conditions are avoided. However, in a distributed memory parallel machine, this strategy can not be used. When a message passing program is run, multiple independent processes are started in different nodes. In this case, the modification of parameters requires a specific protocol to communicate the data from the user interface to the parallel simulation. Therefore, user-triggered events cause the sending of messages to the processing nodes. This communication must be synchronize the execution of the simulation in the nodes in order to correlate the user interaction and the parallel execution.

A suitable way to model this process is by means of finite automata. Each state transition of the automata corresponds to a message from the user interface. The communication protocol defines all possible state transitions of the automata. A simplified model is depicted in Fig. 3. Initial state $C$ represents the starting of a new simulation that can be changed to the execution state $E$. The $E$ to $C$ transition is used to interrupt the current simulation and to start a new one. A transition to the state $S$ causes the simulation to end. The $adjust$ transition allows to change parameters during execution, for example temperature and pressure, while the $configure$ transition deals with "static" parameters, that can be configured only before the simulation starts, as the number of particles or spatial domain. In this way, the complete system models the interaction between an user interface executed in a terminal-provided node and a set of finite automata being executed in the processing nodes (see Fig. 4).
5. The proposed approach

This work proposes a finite automata approach to model interactive MD simulations in distributed memory parallel machines. The ADKS simulator was taken as a case study. The current work presents a parallel version of the ADKS. Its simulation engine was parallelized with the MPI communication library. As in the sequential version, the MOTIF library was used for the visualization. The parallel performance was evaluated for simulations without and with visualization.

In face of the ADKS cell subdivision algorithm, the spatial decomposition approach was used for its parallelization, being equal sets of cells assigned to the processors. For 2D domains, it is more efficient to divide the domain in 1D bands (Fig. 7). Such strategy minimizes communication since each processor has to take into account particles interactions with at most only two other processors. A generic processor $P_i$ will communicate particle coordinates to processor $P_{i-1}$ at one side of its band and to processor $P_{i+1}$ at the other side. A square division of the 2D domain would imply in each processor communicating with much more processors. In a similar way, a 3D domain would be split in 3D slices. This optimizes performance on high latency architectures like PC clusters.

A key issue is how to deal with the interaction between particles in the border of the bands. One possible approach is to perform redundant calculations on each band. This requires the mutual sending of border particle coordinates between bands. Each processor sends updated coordinates of its border particles to neighbouring processors. In the beginning of the simulation all processors have the coordinates of all particles of the system. At each timestep, force calculation is performed for the particles of each processor, including the interaction with border particles of neighbouring processors. Then, resultant forces are computed for each particle and a position check is performed, to update the list of particles of each cell. Finally, border particles are identified and corresponding coordinates are communicated.

Performance results for the ADKS parallel simulation engine decoupled from the visualization are presented in Table 1 and Fig. 5, where $p$ denotes the number of processors. In these tests, a system of particles corresponding to a micro-canonical ensemble (steady energy) is simulated assuming the Lennard-Jones potential. Execution time is given in seconds for 1000 timesteps considering systems with different number of particles, using a cluster with AMD Athlon XP2000 1.67 GHz processors and the Fast Ethernet network. Efficiency has a tendency to increase for larger number of particles, but is limited due to the redundant force calculations of particles in cells that are in the border between processors.

Afterwards, the parallel version of the simulation engine was coupled with the MOTIF-based visualization. Image is updated at every 10 timesteps, but different update rates could be selected. Performance results are presented for the same number of particles in Table 2 and Fig. 8 ($p$ is the number of processors). It can be noted the overhead in the execution time caused by the visualization, as particle coordinates are sent from processing nodes to the visualization node. This is not only due to the communication, but also to the pack/unpack of data that is not contiguous in memory. Execution times for image update at every 5 timesteps, not presented in the tables and graphics, were higher, as it would be expected. For instance, a simulation of 130899 particles and 1000 timesteps using 8 processors, took 57.04 seconds (10 timestep update), but 77.48 seconds (5 timestep update). A single-timestep update would be unacceptable, since it took 133.61 seconds (a speed-up less than 2 for $p = 8$). However, such an update rate double the execution time for the sequential ADKS.

In order to allow a modular approach, the PyMPI [12] Python extension is proposed for a further parallel version

<table>
<thead>
<tr>
<th>$N$</th>
<th>2p</th>
<th>4p</th>
<th>8p</th>
<th>2p</th>
<th>4p</th>
<th>8p</th>
</tr>
</thead>
<tbody>
<tr>
<td>32857</td>
<td>1.61</td>
<td>2.73</td>
<td>4.24</td>
<td>0.81</td>
<td>0.68</td>
<td>0.53</td>
</tr>
<tr>
<td>58855</td>
<td>1.66</td>
<td>2.74</td>
<td>4.04</td>
<td>0.83</td>
<td>0.69</td>
<td>0.51</td>
</tr>
<tr>
<td>91487</td>
<td>1.71</td>
<td>2.91</td>
<td>4.08</td>
<td>0.85</td>
<td>0.73</td>
<td>0.51</td>
</tr>
<tr>
<td>130899</td>
<td>1.62</td>
<td>2.89</td>
<td>4.76</td>
<td>0.81</td>
<td>0.72</td>
<td>0.60</td>
</tr>
</tbody>
</table>

Table 1. Parallel performance of the ADKS simulation engine (AMD XP2000 cluster).
of the ADKS. This extension automatically will provide user interaction with the finite automata at the processing nodes.

Python is an interpreted, object-oriented programming language, that has gained popularity because of its clear syntax and readability. Python is said to be relatively easy to learn and portable to a number of operating systems. Its source code is freely available and open for modification and reuse. Python offers dynamic data type, ready-made class, and interfaces to many system calls and libraries. It can be extended, using the C or C++ languages, thus it can be used to integrate components in a modular approach. The pyMPI Python extension enables users to write distributed, parallel programs based on MPI message passing primitives.

A PyMPI sequential version of the ADKS code and a PyMPI parallel version with no visualization are already implemented, but are not shown in this work. An example of a PyMPI sequential session is as follows, including the ADKS library import, construction of a simulation object (simulate), selection of grain boundary conditions (boundary), execution of 150 timesteps (nstep) for the simulation and image display (show) of the final state of the system of particles, shown in Fig. 6. Thus, it is possible to interactively modify the simulation “on the fly”, as well as to import different modules.

```python
>>> from adks import *
>>> s = simulate()
>>> s.boundary()
>>> s.nstep(150)
>>> s.show()
```

6. Final remarks and conclusions

Some alternatives have been proposed for providing interactive simulations. The Falcon system [3], composed of a set of tools, libraries and user interfaces to execute simulations, visualize data, steer the simulation, etc. Its architecture is based on a steering client, that provides an user interface, and a steering server that controls the simulation engine. The simulation code is instrumented by means of sensors. Each sensor is composed by a data structure and a routine that is inserted in the point of the code to be monitored.
Information gathered by the sensors is stored in queues during execution time in order to be processed and visualized. These sensors can be enabled or disabled in order to tune the degree of monitoring and interference on the simulation code. A thread-based version of this system was implemented in 1995 and a further PVM version was planned.

An alternative, shown in [2], is based on the SPaSM (Scaleable, Parallel, Short-Range Molecular Dynamics) code. This system coordinates simulation, visualization, and analysis. It employs extensible scripting languages such as Python and Tcl/Tk. The Python-compatible builder allows the inclusion of software modules and extensions. In this approach, Python provides an environment that coordinates the high performance MD code, the SPaSM library, the visualization and analysis tools. The Python interpreter was modified in order to be executed on every node of a distributed memory parallel machine. User-input is restricted to a single node that sends messages to the remote Python processes using the MPI library. This allows the development of interactive programs as Python modules, written in C and MPI. New modules can be added in order to provide more functionality. An example is a graphics module that allows parallel rendering. There is a small performance penalty due to the use of a multilayer software environment.

The proposed alternative is to have Python processes executed in every node and controlled by user inputs at the terminal node by means of the pyMPI environment [12], that is an open source extension to Python. The parallel MD code must be rewritten as a set of library routines in order to be loaded and executed in this environment. Specific modules can be included for data analysis and visualization. The use of pyMPI seems feasible and extends all the Python functionality to a distributed memory parallel environment. The Falcon system would require extensive instrumentation of the code, while the SPaSM extension employs a proprietary Python extension.

The current work employs the ADKS simulator as case study. This code was parallelized using MPI only. It was executed on a distributed memory machine, a PC cluster. Performance results are shown for simulations without and with visualization. The analysis of how visualization affects the performance of the simulation is central.

A pyMPI version of the parallel ADKS simulation engine was also implemented and tested. Performance results are equivalent of the non-pyMPI version, since the simulation engine is the same, but are not shown in this work.

Next steps are underway and include to rewrite some parts of the ADKS code in order to provide better modularity, to implement a new parallelization scheme in order to avoid redundant calculations in the borders between partitions of the domain assigned to processors and to finish the interactive implementation of the pyMPI version.

In order to check the influence of the communication overhead in the performance, the ADKS simulations were executed, with visualization, in a parallel machine with a faster network. It was employed an ItautecR/AD cluster, based on Intel Pentium III Xeon 1.26 GHz dual processors and a Gigabit Ethernet network. Execution times were slightly higher (Fig. 9), in comparison to the AMD XP2000 cluster, but speed-up and efficiency values, shown in Table 3, demonstrate the convenience of a high speed network for this kind of simulations.

An interesting point is that number of particles in an interactive simulation may be restricted by data communication and visualization issues, rather than by the simulation itself. The image rendering in interactive simulators is usually performed at the user-interface node, but number of particles in the order of millions may require a parallel rendering, adding complexity to the software. As future work, the simulation steering capability could be com-

### Table 2. Parallel performance of the ADKS with visualization (AMD XP2000 cluster).

<table>
<thead>
<tr>
<th>N</th>
<th>2p</th>
<th>4p</th>
<th>8p</th>
<th>2p</th>
<th>4p</th>
<th>8p</th>
</tr>
</thead>
<tbody>
<tr>
<td>32857</td>
<td>1.77</td>
<td>2.60</td>
<td>3.45</td>
<td>0.88</td>
<td>0.65</td>
<td>0.43</td>
</tr>
<tr>
<td>58855</td>
<td>1.84</td>
<td>2.72</td>
<td>3.63</td>
<td>0.92</td>
<td>0.68</td>
<td>0.45</td>
</tr>
<tr>
<td>91487</td>
<td>1.81</td>
<td>2.79</td>
<td>3.64</td>
<td>0.90</td>
<td>0.70</td>
<td>0.46</td>
</tr>
<tr>
<td>130899</td>
<td>1.74</td>
<td>2.75</td>
<td>3.86</td>
<td>0.87</td>
<td>0.69</td>
<td>0.48</td>
</tr>
</tbody>
</table>

### Figure 8. Execution times of the ADKS with visualization (AMD XP2000 cluster).
implemented with computational steering, in order to provide better load balancing. For example, dynamic processor allocation could be exploited using the MPI-2 standard.

7. 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. Author E. R. Rodrigues is supported by a MSc grant of FAPESB, The State of Bahia Research Foundation. The authors are grateful to Dr. Enzo Granato and Dr. Leonel F. Perondi from the Applied Laboratory for Sensors and Materials (LAS/INPE) for the helpful support and suggestions.

References