Large scale simulation of parallel molecular dynamics

Pierre-Eric Bernard
Numath, INRIA Lorraine, BP 101,
54600 Villers les Nancy, FRANCE
Pierre-Eric.Bernard@loria.fr

Thierry Gautier and Denis Trystram
APACHE group, LMC-IMAG, BP 53X
Grenoble, F-38041 Cedex 9, FRANCE
Thierry.Gautier@inrialpes.fr, Denis.Trystram@imag.fr

Abstract

This paper aims to describe the implementation of TAKAKAW*, an efficient parallel application for the simulation of molecular dynamics designed to handle large proteins in biology. The implementation is based on the multi-threading parallel programming environment, called* ATHAPASCAN [1] *which allows to implement and evaluate easily several load-balancing strategies. Some experiments run on one of the largest molecule ever simulated (an hydrated β-galactosidase with 413039 atoms) show the interest of such a parallel programming environment.*

keywords. Load balancing - Threads - Parallel Molecular Dynamics.

1 Introduction

Simulations of molecular dynamics is an important challenge of today. Biologists started recently to study the behavior of large molecular systems consisting of several thousands of atoms with the help of parallel computers [15, 6, 2]. The computational complexity of such systems increases considerably with the molecule size.

Part of molecular dynamic application is to simulate the behavior of multiple-particles systems via numerical simulation techniques. The principle is to compute time series of positions and velocities of the atoms by integrating Newton's equation of motion. At each step of molecular dynamic application, the forces between atoms have to be computed. The computation of non-bonded forces, the Van der Waals and electrostatic (Coulomb) forces between charged atoms is the most consuming portion for a typical simulation [22, 5, 21, 15]

The non uniform distribution of the atoms in space produces highly irregular computations, the main challenge is to provide a high performance implementation that scales well as the size of the problem increases. Many parallel implementations of molecular dynamic application have been done. Most implementations consider static load-balancing with stop and restart in case of important work-imbalance at execution time. Only a few consider dynamic load-balancing [15, 4].

TAKAKAW was designed to be able to simulate large systems of atoms with the van der Waals and Coulomb interactions as well as geometric interactions [2, 6] using Newtow's equation of motion. The experiments on one of the largest molecule ever simulated (an hydrated β-galactosidase with 413039 atoms) is reported in this paper,

The organization of the paper is the following. We first recall and discuss the common approaches used to parallelize molecular dynamics. Then, we present the parallel method and we briefly present the parallel programming environment on which TAKAKAW has been implemented. We give the keypoints of our implementation that improve efficiency. In section 3, we detail the algorithms of our parallelization. We especially focus on the load-balancing problems. Finally, some experiments are presented and analyzed on the CRAY T3E until 256 processors for studying the dynamic of a large system of atoms in the area of biology.

2 Model and parallel methods

2.1 General description

Molecular dynamics (MD) aims at studying the dynamic behavior of multiple-particles systems via numerical simulation techniques. This generic method is widely used for simulating the properties of liquids, solids and gas. It is also used to study mechanical and structural properties of proteins and other biological molecules [6] [17] [2]. The principle is to compute time series of positions of the atoms by integrating Newton's equation of motion. Within this model, an atom is considered to be a charged point mass of a given type (oxygen, carbon, ...).

[1] ATHAPASCAN is part of the research project APACHE supported by CNRS, INPG, INRIA and UJF

The numerical solution of these equations is usually computed by an iterative integration loop from the positions of the atoms and their velocities. Initial positions and velocities are given. It is necessary at each time step to compute the forces between each atom of the system. The system is essentially influenced by two types of forces:

- intra-molecular geometrical forces which model the vibrations along covalent bonds, the bending vibrations between two adjacent bonds, and the torsional motions around bonds.
- usual non-bonded interaction forces of Coulomb and Van der Waals. They are involved between all pairs of atoms in the whole system.

Solving the previous set of equations can be expressed in several steps nested inside a global integration loop. We sketch below the principle of the general solution method:

```
for all time steps do
   compute intramolecular forces ;
   compute non-bonded forces ;
   integrate Newton's equation of motion ;
end do;
```

Computing exactly the non-bonded forces may require to calculate the non-bonded interactions between each atom in the system with every other atom, giving rise to $O(N^2)$ evaluations of the interaction in each time step. In practice the non-bonded forces consume the most significant part of the computation time. It takes more than 90% of the total [22, 5, 21, 15]. For large systems, it becomes necessary to avoid the explicit computation of all non-bonded interactions. However, MD application usually truncates calculation of the non-bonded forces at a fixed *cutoff* radius. The non-bonded interactions decrease rapidly with increasing distance between atoms. Then, it is possible to neglect the interactions between two atoms separated by more than a certain radius (called the *cutoff* radius). This means that an atom has only non-bonded interactions with the atoms which are in a sphere with a radius equal to the *cutoff* [6, 2].

The *cutoff* method reduces considerably the computations. However, our objective is to simulate the dynamic of systems constituted by at least several tens of thousands atoms for more than one hundred of thousands iterations (this represents about 100 ns of the biological simulation). Although the *cutoff* approximation does not allow to take into account all the electrostatical phenomena. However, it is common to perform a series of integration steps with *cutoff* between full-range iteration steps. Moreover, it can be used as a basic bulk for many other methods (like multipole methods (PFMA) [3]).

There exist essentially three kinds of numerical methods for solving the molecular dynamics equations, namely, the one which computes all the interactions, the PFMA, and the *cutoff* approximation methods. These general methods are discussed in Plimton [20].

2.2 Parallel methods

Parallel implementations for large scale simulation in MD have to distribute the data among the processors with the lowest possible communication cost.

We can essentially distinguish three ways to parallelize MD application.

- **Atom decomposition.** The atoms and the corresponding non-bonded interactions are distributed among the processors. All the coordinates of the atoms are exchanged before computing the interactions. Then, they exchange the computed forces and calculate the new positions of the atoms. These methods are characterized by reference to a collective communication pattern (total exchange) which makes it rather inefficient. However, this algorithm has been widely used because it is easy to implement [17] especially on shared-memory multiprocessors [19].
- **Force decomposition.** These methods are based on a block splitting of the matrix representing all the combinations of atom pairs. They avoid the previous expensive collective communication but, do not take into account the locality of data needed for calculating intermolecular forces. They are not scalable. They also require a number of processors equal to a perfect square. For this reason, such methods are difficult to implement in practice [12].
- **Spatial decomposition.** These methods correspond to a geometric decomposition of the domain. They are often used for expressing parallelism with fine granularity. Each part of the physical domain is assigned to a processor. The main characteristic is that the atoms are not assigned to a given processor, but they are allowed to move from one processor to its neighbors [8, 16, 13].

Complexity of these three methods are summarized in table 1.

Method	Arithmetic cost	Communication cost
Atom decomposition	O(N/p)	O(N)
Force decomposition	O(N/p)	O$\left(N/\sqrt{p}\right)$
Spatial decomposition	O(N/p)	O(N/p)

Table 1. Complexities of the three classical methods to parallelize on p processors a MD simulation with N atoms.

In this paper, we are interested in the last method (also called *link-cell* method [20]). Due to its complexity, it will scale well on large parallel architectures. The data decomposition enforces locality of access to the neighbors of one atom.

2.3 Overview of the program

The input of MD application is the description of a system of particles in a thermodynamic state, the output is a time series of thermodynamic states.

The principle of the parallel MD application based on spatial decomposition, is to bin the atoms into 3-dimensional boxes of side greater or equal than the *cutoff* radius. In this case, the interactions are limited to the 26 neighbor boxes, that bound the communication cost.

Our parallel MD application subdivides the program into smaller tasks associated with boxes: the tasks for integration of equations of motion, the tasks for computation of non-bonded interactions for each boxes and pair of box, the tasks for evaluation of geometric interactions *etc.*. Dependency constraints between these tasks are based on a data flow analysis of the algorithm. Additional constraints on tasks are defined in order to preserve the locality of data access: for instance, the tasks that compute geometric forces and non-bonded interactions for the i-th box are mapped onto the same processor. Then, the cost of each task is estimated using the number of atoms in each box, and a mapping algorithm is used to compute the initial distribution of the tasks and the boxes among the processors.

Step after step the integration of equations of motion introduces a drift of the work-load of the processors: some runtime adjustments have to be done within a specific period (see section 3.2).

From the implementation point of view, the tasks are executed by light weight processes (or threads discussed in the next section). A thread has its own control flow and executes concurrently with other threads. On each processor of parallel distributed machine, TAKAKAW maintains different threads: threads for local computation (one thread for non-bonded interactions, an other one for other forces and integration), threads for communication (two threads per node for each other processors, a first one that waits to receive position and sends local evaluation of forces, and the second one that is symmetric to the first). Figure 1 shows a visualization of the activity of the multi-threaded program.

2.4 ATHAPASCAN: a multi-threading parallel programming environnement

Given a parallel algorithm, a good implementation requires that all potential parallelism inside a parallel machine have to be used: parallelism across different processors, parallelism between the communication and the computation. Light weight processes (shortly *threads*) have been shown as a good tool to efficiently use this different levels of parallelism. They are useful to manage dynamic creation of parallelism (new threads of control), due to their low costs of creation and context switch. They are also widely used for masking communication cost (latency), or to asynchronously send of receive messages. The local multiprogramming of the processors allows to schedule efficiently the threads corresponding to each service.

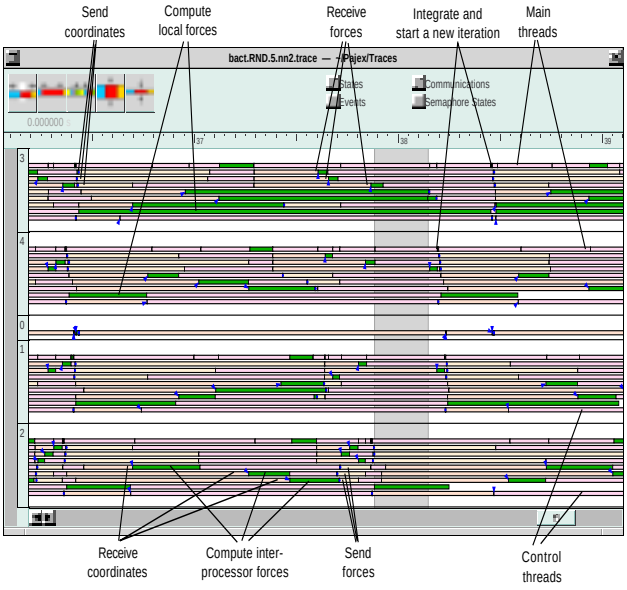

Figure 1. Principle of the execution scheme on one processor for one iteration.

The APACHE project aims at developing a parallel programming environnement, called ATHAPASCAN, for high performance computing [1]. An ATHAPASCAN program is a set of nodes interconnected by a network. A node is a Unix process: each node has its own address space shared by several threads of control. A node can be mapped onto a symmetric multi-processors (SMP). ATHAPASCAN provides several methods for communication between threads (shared data with synchronization features (mutex, semaphore), message passing, remote memory operations -read/write-). Basically, all operations from/to a remote node are asynchronous to overlap some communication cost by computation.

Instrumentation of the ATHAPASCAN runtime support [10] is used to do post-mortem performance evaluation of an execution of any ATHAPASCAN program. A visual tool permits to analyze each specific step of a parallel execution (figure 1).

The runtime support ATHAPASCAN provides efficient implementation [7] of thread and communication functionalities on various distributed platforms (IBM SP2, Cray T3E, SGI Origin 2000). ATHAPASCAN is built on top of the standard interfaces: MPI and POSIX-threads.

Figure 1 focuses on the execution scheme on four processors for one iteration[2]. This tool permits to have access to a pictural behavior of the program but also for quantitative measurement as specific times (communication, computation) or statistic quantities. This figure has been draw using a random mapping of tasks (discussed in section 3.1). It shows the relatively poor performances and work-imbalance of the execution.

3 Load-balancing issues

The load-balancing scheme in TAKAKAW is composed by an initial mapping of data among the processors and of runtime adjustment of the work-imbalance.

3.1 Initial data distribution

The initial allocation is computed by an algorithm based on a recursive bisection for distributing the boxes onto the processors. As we have already mentioned, the domain has been decomposed into cubic boxes whose side is larger or equal to the cut-off radius. These boxes may contain different numbers of atoms depending on the biological structure. The computational load of non-bonded interactions is subdivided into tasks associated with boxes: the tasks for computation of non-bonded interactions for each box and for each pair of neighbor boxes. Moreover additional constraints enforce the tasks to be mapped on the same processor as the one of its associated boxes. The recursive bisection method allows to capture the trade-off between a good repartition of the number of atoms (which is proportional to local computations) and a relatively small communication cost [18].

The principle of this algorithm is recalled on figure 2. In order to validate this initial allocation scheme, it has been compared to a pure load balancing strategy which does not take into account any communication: a greedy algorithm based on the well-known LPT rule (which stands for Largest Processing Time first [11]). The tasks that compute non-bonded inter-actions associated to one box are first sorted in decreasing order of load (i.e. the number on non-bonded interactions they have to compute). Then, they are allocated to the processors by chosing the less loaded one. And then, according to previous mapping constraints, other

[2]For clarity of the figure, we put the master thread of the application onto a specific processor with label 0. Usually it is mapped onto one of the slave processors.

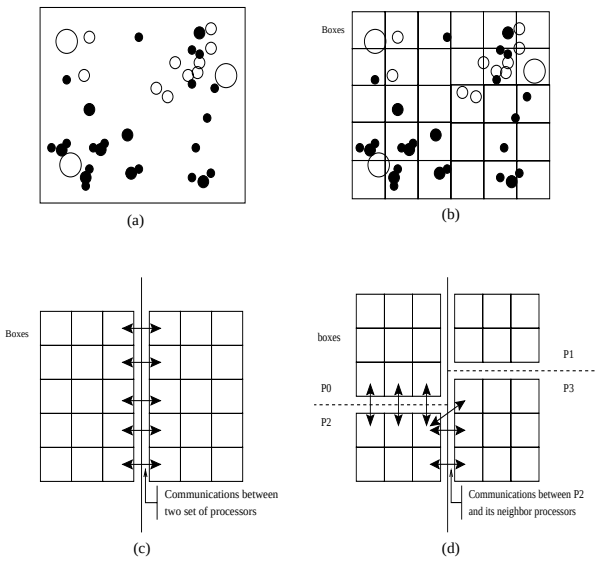

Figure 2. Recursive partition (2 levels) of the box set in 2 dimensions.

tasks are mapped in a same manner. It is known to be efficient for very large grain parallelism [9].

3.1.1 Impact on performances

Some experiments were run on a 32 nodes IBM SP parallel machine in order to compare static load balancing policies (LPT and recursive partition). The results given in figure 3

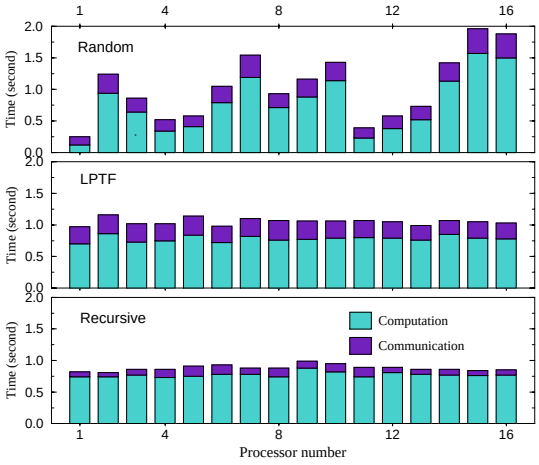

Figure 3. Work-imbalance of one iteration over 16 processors, using the different mappings. The biological system is a dioxolane-gramicidine with 11615 atoms.

concern one iteration. As expected, the algorithm based on

the recursive bisection outperforms LPT. We run the code of the best static policy recursive bisection in order to study the evolution in time. Figure 3 shows that the load balancing behaves well but the performances decrease when the system evolves. The next section will investigate the potential benefit of using dynamic correction of the load repartition.

3.2 Dynamic adjustment

3.2.1 Lazy task migration

In our MD application, the movement of atoms causes irregular repartition, and this phenomenon becomes more and more critic as the time evolves (as shown in the left part of figure 4, using the recursive bisection algorithm).

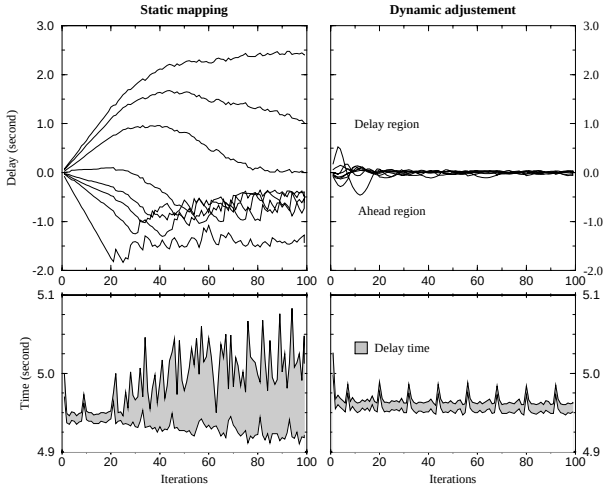

Figure 4. Comparison of execution times between static mapping (left part) and dynamic adjustment (right part) on 100 iterations for the simulation of an actual system with 35349 atoms. The upper part of the figure shows the delay or the ahead for each of the eight processors from the average time of an iteration. The down part of the figure shows the average times for each iteration decomposed in computation time (bottom line) and real time (top line). The region in grey is the delay time for each iteration.

After a phase of transition, the imbalance of work-load is bounded by the slow processor that reaches the end of iteration. We only develop some local adjustments of the load: only tasks that do not require communication of data are moved, depending on the delay from the estimation time to complete an iteration. From a good initial distribution of work, it allows slight imbalance without lost of efficiency (right part of figure 4).

The task that computes non-bonded forces between two neighbor boxes could be moved without further communication: such tasks need positions of atoms that are sent even if the task is not moved. Moreover this is the only kind of tasks that could be sent without further communication. The estimation of the time to complete i-*th* iteration uses times of completion of the $(i-2)$-*th* iteration. During iteration $i-1$ the estimations are broadcasted to processors.

The reactivity of the dynamic adjustment algorithm is at least 3 iterations. Using an important number of processors for medium size problems suppresses the improvement of the dynamic adjustment algorithm. This can be explained because the maximum imbalance between processors is smaller using more processors. For instance, the maximum delay between any processors is 3 seconds using 8 processors in experiment report in figure 4. Using 16 processors, this maximum delay decreases until $0.8s$.

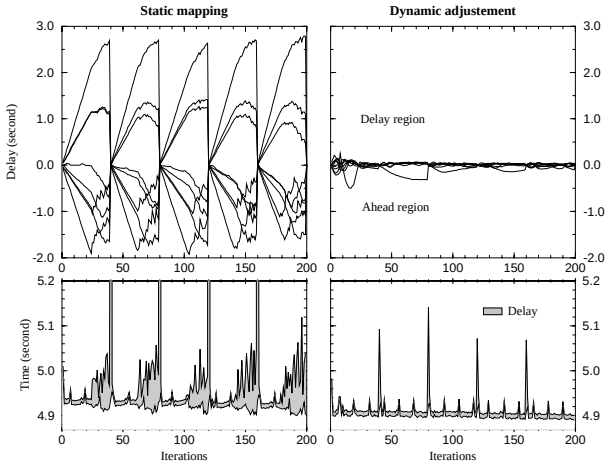

Figure 5. Comparison of execution time between static mapping (left part) and dynamic adjustment (right part) on 200 iterations for the simulation of system of figure 4. A synchronization to update internal data structures (position of atoms) occurs every 40 iterations.

3.2.2 Migration of atoms

The motion of atoms all over the simulation could be more important than the size of a box (even if we consider the size of box extended by a tolerance zone). Two cases may occur: if the atom changes between boxes but stays into the same processor, or if the atoms changes to a neighbor box located in another processor.

Changing boxes of atoms requires to update internal data structures to fast access to their positions, that synchronize all processors. TAKAKAW updates this data structure every several tenths iterations using a distributed algorithm. Figure 5 shows the experiments on the same system as for figure 4 (with 35349 atoms). The left part reports experiments using the initial recursive bisection mapping algorithm and a synchronization every 40 iterations to update the data structure. After each synchronization, the behavior of the simulation is similar to the previous figure 4.

In studied biological systems, the work-imbalance does not vary to much for hundreds of iterations. The initial mapping produced by recursive bisection and the dynamic adjustment are enough for typical simulation of thousands iterations. In case of an important work-imbalance, the simulation is stopped and restarted with a new initial data distribution using the recursive bisection algorithm [15].

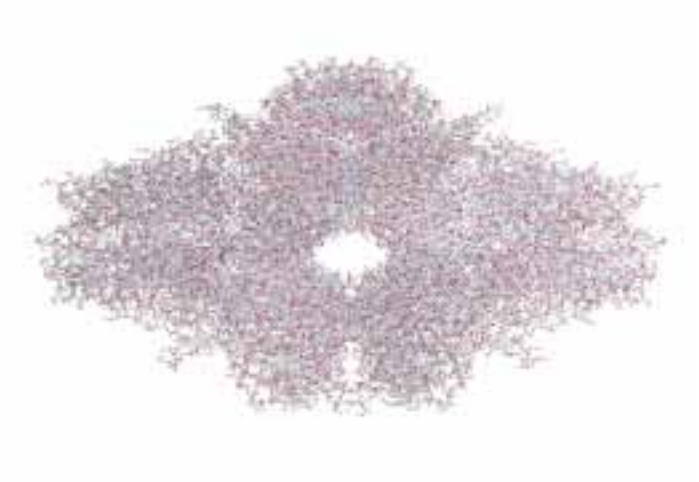

Figure 6. Shape of the β-galactosidase alone (without the solvant) a protein of about 65 240 atoms.

4 Simulation of a β-galactosidase

To show that our application is able to simulate large molecular structures of proteins, we have simulated the movement of the largest protein structure found in the Brookhaven Protein Data Bank: β-galactosidase[14] (figure 6). The protein is immersed into a 100 Å radius sphere of water of about 65 240 atoms. The total number of atoms into the system is 413039 atoms. The size of this system is more than 4 times larger than the size of the systems which can be simulated with the other current applications of molecular dynamics.

4.1 Simulation onto IBM-SP

For this simulation, the *cutoff* radius has been fixed to 10 Å. We started with a small time step of integration of $0.1 \cdot 10^{-15}\,s$ to equilibrate the system. We carried out a first simulation of 300 steps to gradually heat the structure from 50 K to 300 K, rescaling the atoms velocity according to the target temperature every all 5 time steps. Then we led a simulation of 700 steps at a constant temperature of 300 K (we rescaled the velocity every all 10 steps) to well equilibrate the structure.

We then carried out a long simulation of 4 times 1400 steps with a time step of $0.25 \cdot 10^{-15}\,s$ and without rescaling the velocity, to check the energetic stability of the system. This last test represents 35 hours of calculation on 20 nodes of the IBM-SP.

4.2 Simulation onto CRAY T3E

To check scalability of our application, we also performed some experiments on a T3E[3] with a large number of processors. Table 2 gives the average times of an iteration of molecular dynamics for the system of 413039 atoms that contains the β-galactosidaze. We did not show the efficiency because it is not significant. Indeed there is not enough memory on a node of the T3E to allow the computation of a such big system on less than 8 processors. In fact, it needs even 32 processors to have enough memory to generate all the non-bonded interaction lists and to perform computation at full speed. Below 32 processors only a part of the non-bonded list of interaction is generated, according to the size of the free memory on each node. For a task that computes non-bonded interactions, if there is not interaction list available for this task, the application computes interactions between all pairs of atoms inside the boxes associated to the task. It is slower than using a list of the non-bonded interactions, but it uses less memory. On 256 processors, useful computation for the simulation is 82% of the time of an iteration. The 18% remainder are, for a half, the delay time due to imbalance and for other half the computing time to manage distributed data structures (displacement of the atoms over the processors) and the management of the communications.

413 039 atoms (time in second)			
Nb. Proc. (p)	8	16	32
$t_{\text{tot}}(p)$	65.21	27.70	11.69
Nb. Proc. (p)	64	128	256
$t_{\text{tot}}(p)$	5.88	3.07	1.69

Table 2. Average execution time and efficiency of an iteration of molecular dynamics on a T3E. The *cutoff* radius is $10\mathring{A}$.

5 Conclusions

In this paper we have described TAKAKAW, a molecular dynamic application designed for large scale simulations in biology. We have reported some experiments on one of largest molecule ever simulated (a β-galactosidase with 413039 atoms). This simulation has been possible with the design of specialized algorithms for the dynamic load

[3] T3E-750 with 256 processors (EV5.6 375Mhz) and each with 128 Mo of memory

balancing problem (recursive partition and dynamic adjustment), but also by the use of the efficient runtime support (ATHAPASCAN) for parallel multi-threading programming. Moreover this code is portable across different platforms (IBM-SP, Cray T3E, SGI Origin 2000).

Current developments are to improve treatment of the hydrated medium (simulated proteins are surround molecules of water) to a better approximation of the electrostatic interactions. Our dynamic molecular code will be coupled with a code for numerical solution of partial derivative equations applied to electrostatic fields within charges. This is part of SimBio, an INRIA project[4].

Acknowledgement

B. de Oliviera Stein designed the visualization tool Pajé and produced the figure 1. Biologists Y. Chapron and S. Crouzy from CEA[5] are from five years ago at the beginning of a faithful collaboration in development of TAKAKAW.

References

[1] *APACHE Project*. http://www-apache.imag.fr.

[2] A.T. Brünger. *X-PLOR A System for X-ray Crystallography and NMR* . Yale University Press, 1992.

[3] J. Barnes and P. Hut. A Hierarchical $O(N \log N)$ Force-Calculation Algorithm. *Nature*, 324:446–449, December 1986.

[4] P.-E. Bernard, B. Plateau, and D. Trystram. Using threads for developing parallel applications: Molecular dynamics as a case study. In Trobec, editor, *Parallel Numerics*, pages 3–16, Gozd Martuljek, Slovenia, September 1996.

[5] K. Boehmcke, H. Heller, H. Grubmüller, and K. Schulten. Molecular Dynamics Simulations on a Systolic Ring of Transputers. *Transputer Research and Applications 3*, pages 83–94, 1990. Washington DC 1990 NATUG, IOS Press.

[6] B.R. Brooks, R.E. Bruccoleri, B.D. Olafsen, D.J. States, S. Swaminathan, and M. Karplus. CHARMM : A Program for Macromolecular Energy, Minimization, and Dynamics Calculations. *Journal of Computational Chemistry*, 4(2):187–217, 1983.

[7] A. Carissimi and M. Pasin. Athapascan: An experience on mixing mpi communications and threads. In V. Alexandrov and J. Dongarra, editors, *Proc. of 5th European PVM/MPI Users' Group Meeting*, number 1497 in Lecture Notes in Computer Science, pages 137–144, Liverpool, UK, September 1998.

[8] T.W. Clark, R.v. Hanxleden, J.A. McCammon, and L.R. Scott. Parallelizing Molecular Dynamics Using Spatial Decomposition. *Scalable High-Performance Computing Conference (SHPCC94)*, page 95, 1994.

[9] M. Cosnard and D. Trystram. *Algorithmes et architectures parallèles*. InterEditions, Collection IIA, 1993. Translation in english by ITP, 1995.

[10] B. de Oliveira Stein and J. Chassin de Kergommeaux. Pajé, an interactive and visual tool for tuning multi-threaded parallel applications. *International Journal of Software Engineering and Knowledge Engineering*, 1998. Submitted.

[11] R.L. Graham. Bounds for certain multiprocessor anomalies. *Bell System Tech J.*, 45:1563–1581, 1966.

[12] B. Hendrickson and S. Plimton. Parallel Many-Body Simulations Without All-to-All Communication. Technical report, Sandia National Laboratories, Albuquerque, NM 87185, 1994.

[13] P.A.J. Hilbers and K. Esselink. Parallel Molecular Dynamics. *Parallel Computing: From Theory to Sound Practice*, pages 288–299, 1992.

[14] R.H. Jacobson, X.-J. Zhang, R.F. DuBose, and B.W. Matthews. Three-dimensional structure of β-galactosidase from E.Coli. *Nature*, 369:761–766, 1986.

[15] L.V. Kalé, M. Bhandarkar, and R. Brunner. Load balancing in parallel molecular dynamics. In H. Simon A. Ferreira, J. Rolim and S.-H. Teng, editors, *5th International Symposium IRREGULAR'98*, number 1457 in Lecture Notes in Computer Science, pages 251–261, Berkeley, California, USA, August 1998.

[16] V. Lakamsani, L. N. Bhuyan, and D. Scott Linthicum. Mapping Molecular Dynamics Computations on to Hypercubes. *Parallel Computing*, 21:993–1013, 1995.

[17] S.L. Lin, J. Mellor-Crummery, B.M. Pettitt, and Jr. G.N. Phillips. Molecular Dynamics on a Distributed-Memory Multiprocessor. *Journal of Computational Chemistry*, 13(8):1022–1035, 1992.

[18] R.J. Lipton, D.J. Rose, and R. Tarjan Endre. Generalized nested dissection. Technical Report CS-TR-77-645, Stanford University, Department of Computer Science, 1977.

[19] J.E. Mertz, D.J. Tobias, C.L. Brooks III, and U.C. Singh. Vector and Parallel Algorithms for the Molecular Dynamics Simulation of Macromolecules on Shared-Memory Computers. *Journal of Computational Chemistry*, 12(10):1270–1277, 1991.

[20] S. Plimpton. Fast Parallel Algorithms for Short-Range Molecular Dynamics. *Journal of Computational Physics*, 117:1–19, 1995.

[21] R.Trobec, I.Jerebic, and D.Janezic. Parallel Algorithm for Molecular Dynamics Integration. *Parallel Computing*, 19:1029–1039, 1993.

[22] M. Sprik. Running Molecular Dynamics Code in Parallel on a Cluster of Workstations. *SPEEDUP*, 6(2):57–61, 1992.

[4] http://www.iecn.u-nancy.fr/Textes/Recherche/Equipes/Numath/SIMBIO/index.html

[5] BMC/DSV/DBMS CEA-Grenoble.