PROTOMOL: A Molecular Dynamics Framework with Incremental Parallelization

T. Matthey[*], *J.A. Izaguirre*[†]

1 Introduction

Molecular dynamics (MD) for a classical unconstrained simulation of bimolecular systems requires the solution of Newton's equations of motion. At each step, one evaluates the contribution of interacting forces, and these are applied to the system using a numerical integrator. The most computationally expensive part is the force evaluation among atoms. MD is a challenging application because the time scales are long (processes of interest are in the order of microseconds to seconds, and the time steps are in the order of femtoseconds), and also because the system sizes are very large (thousands to hundreds of thousands of atoms). The time scale problem is typically addressed by using multiple time stepping integrators, whereas the second problem may be addressed through the use of parallel processing.

In this paper, we evaluate the design of PROTOMOL [7], a framework for MD that uses encapsulation and generic programming to provide an extensible component platform for parallel algorithms for MD. The emphasis on design and extensibility distinguishes PROTOMOL from other excellent MD programs, such as GROMOS [15], Amber [18], CHARMM [2]. A program with similar goals, which provided the initial inspiration for PROTOMOL, is NAMD2 [8]. However, NAMD 2's design goal is primarily high scalability, which forces the algorithm developer to consider the parallelization scheme of the program from the outset.

We present an approach that hides parallelization details on some objects, and which thus allows for the incremental development of parallel modules within PROTOMOL. Our

[*] Department of Informatics, University of Bergen, Norway,*matthey@ii.uib.no*
[†] Department of Computer Science, University of Notre Dame, *izaguirr@cse.nd.edu*

goal is to make parallel optimization simple and flexible enough, so that the parallelization can be made transparent for the application developer, while we rely on generic programming techniques to achieve high performance. This approach has been proven by a number of projects such as POOMA [5] project, the MTL [14], Blitz++ [17], and similar efforts.

Our *incremental parallelization* scheme provides generic parallelization mechanisms and facilities for clusters with moderate number of nodes, e.g. up to 64 nodes. The idea is that the applications developer may start with a sequential implementation of his or her algorithm, while transparently benefiting from already parallelized parts in the framework; later he or she may parallelize the whole new algorithm. Our goal is to have a platform that may extend eventually from a few processors to large scale systems, and which may include the use of multi threading and distributed message passing.

Our design is based on a force decomposition that admits different work distribution approaches, from static to dynamic distribution using a master-slave paradigm. The paper is organized as follows: First, we present the design of PROTOMOL, and compare it with state-of-the-art MD programs. Then, we describe its parallelization, presenting results of some benchmarks: using a 128 node Origin2000, we get a speedup of 24 on 32 nodes for a 92224 atom system, using a master-slave paradigm, and a speedup of 22 on 32 nodes for the same system, using a static distribution. We conclude that our approach to incremental parallelization is flexible, and shows reasonable scaling. In the last section, we propose a refinement of PROTOMOL that will allow us to improve the performance, with the future goal of accommodating large scale systems and implementations combining multi threading and message passing.

2 Design of PROTOMOL

PROTOMOL is a component framework for MD simulations. It has generic components to represent MD integrators (allowing for arbitrary chains of multiple time stepping algorithms), generic forces, both bonded and non-bonded, and generic boundary conditions. It provides mechanisms for the efficient evaluation of all the computational components. It relies heavily on templates and inlining to achieve both abstraction and high performance. Fig 1 gives an overview of PROTOMOL.

The integration – one time step – consists basically of the evaluation of forces, and the updating of positions and velocities. The multiple time stepping integrators can be set in a chain, so that particular forces (or ranges of forces) are assigned a given time step. The user can define new integrators by allocating some force evaluations for each level.

3 Incremental Parallelization of PROTOMOL

We use an object that parallelizes a given computational work by distributing the work and collecting the contributions. The implementation uses MPI and one-sided communication features of MPI-2 to optimize communication.

To meet these computational demands there are several sequential optimization approaches and four basic strategies [16] for parallelization: (a) The master-slave approach, which allocates work among slaves and has both communication and load balancing difficulties. (b) The atom decomposition, which is based on data replication; it is an easy but

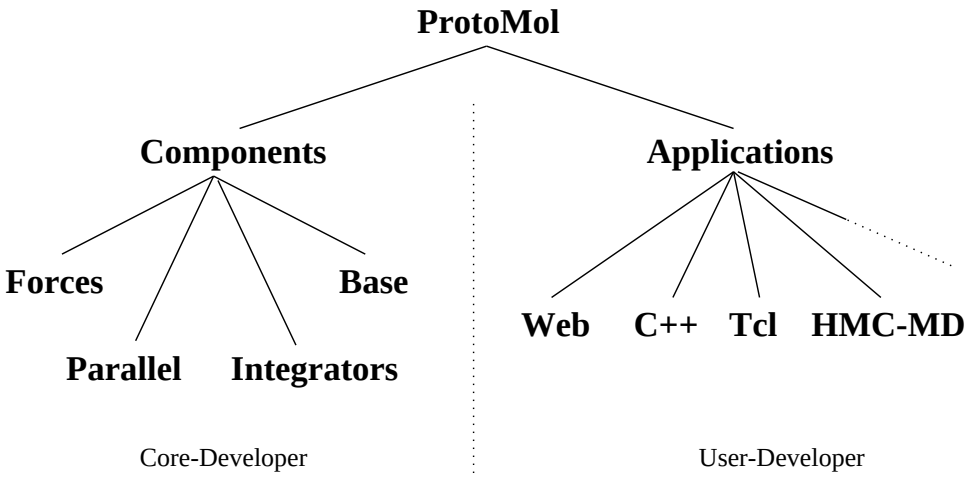

Figure 1. *The Framework.*

memory expensive approach with poor scaling due to global communication. Programs using this decomposition include GROMOS [15], Amber [18], CHARMM [2], and an early version of EGO. (c) The force decomposition, which involves either force matrix or systolic loop methods. It scales better than atom decomposition by reducing communication costs through the use of a block-decomposition, as in LAMMPS [13] and CHARMM [6], and as discussed in reference [12]. (d) Spatial decomposition, which uses linked lists and scales very well, but it is more difficult to write and extend. It is used in ddgmq [3], SIGMA, NAMD2 [8], PMD [19], EulerGromos [4] and it is evaluated in references [1, 9].

3.1 Work Distribution and Data Transfer

Work distribution is performed by the master, through a short message (work-command). Slaves retrieve the required data independently, carry out the work, and notify the master when finished, see Fig. 2a. Data may either be centralized, distributed or replicated, depending on the parallelization implementation desired. To distribute data among the slaves we use proxies. The slaves exchange data for the assigned work independently (Fig. 2b) and without any explicit synchronization via proxies; the owner of data makes its wanted data globally accessible. Access of data is transparent, as in the sequential case. Caching and smart reduction of contributions are also hidden in the proxies. Communication is based on MPI and features of one-sided communication MPI-2 [9, 10].

3.2 Load Balancing

We tested both dynamic and static load balancing. Dynamic load balancing can be done either on a local or on a global level, where the global load balancing is in general better, but it uses global synchronization. Load balancing becomes more important for large number of nodes [8, 11]. For our purpose, with a moderate number of processors, we can handle

load balancing by one master globally, where as for massively parallel processors we need a hierarchy of or several masters.

Static load balancing can be appropriate, when ever the force work load – at least for the most expensive ones – can be estimated and distributed equally among the nodes without any run time information. The advantage is that we do not need to explicit communicate work-commands, but it clearly will not scale for large number of nodes or for small systems.

3.3 Range Computation

With the data in place on a local node, the work can be carried out. The actual assigned work (part) is defined by a range – or more generally by a selection, e.g. representing a cell – of atoms. This requires range computation capabilities in the functions to be parallelized. Range computation should be simple to introduce as long as we guarantee that all the data is in place. The idea is that the computation function only has to be extended to handle ranges or selections of atoms. However, the application developer does not need to add selection or range capabilities as long as the work or function does not need to be executed in parallel.

During a time step the master sends a short work-command (Fig. 2b) telling what and on which atoms the work should be carried out. The slave gets the needed data via the proxies and executes the work. As soon the slave finished, it notifies the master and waits for further work. The contributions or results are sent back to master by the proxies typically at least at the end of a time step. We have to consider that we may occasionally rearrange the atoms for computations, wherever the needed data objects are not contiguous, to reduce the data transfer.

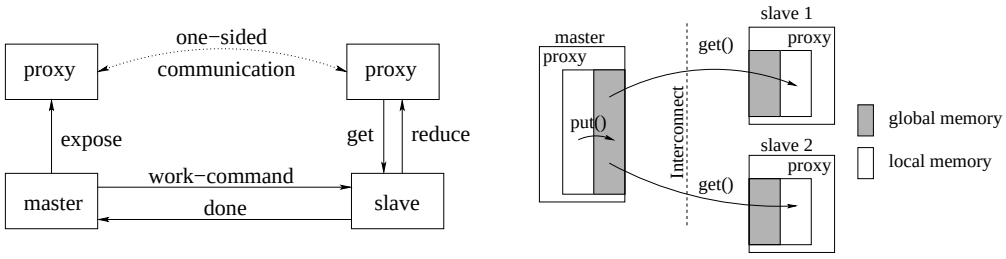

Figure 2. *(a) The master-slave paradigm, (b) The data (memory) exchange between the master and the slaves.*

3.4 Implementation and Evaluation

We parallelized the force group class, by distributing different forces to different nodes. For example, a simulation may evaluate several bonded forces (such as bond and angle forces) and several nonbonded forces (such as Coulomb and Lennard-Jones). Some of these forces are themselves evaluated in parallel, provided the force classes support range computation. Ranges of atoms are distributed transparently among available computational nodes, according to a particular load balance strategy, as explained below.

It is natural to parallelize the updating and force computation together such that communication is minimized. Every node is assigned a range of atoms in which to update positions and velocities. Each node makes its force contribution public using one-sided communication in MPI-2. Then all nodes perform reductions for their assigned range. After a local update of positions and velocities, the nodes make this data public.

The proxies keep track of the state of the data (valid, unavailable, distributed, etc.) The validation of data is done at this level to allow the transfer of larger chunks of data, for higher performance.

We tested a static load balancing. Sequential forces are distributed equally among the nodes. Parallel forces determine by themselves which range they have to compute. For the full nonbonded forces we divide the force matrix into p equally areas, where p is the number of nodes. For cutoff nonbonded forces we assign sets of cells to the nodes.

We ran our framework for a moderate example with 14281 atoms evaluating Coulomb forces and Lennard–Jones forces with a cutoff of 7 Å and bond, angle and dihedral forces on an Origin2000. The speedup for 8 nodes is about 6.5 and for 32 nodes around 16. The reason for the poor scaling is load unbalance.

Then, we tested a simple dynamic load balancing, which is based on the master-slave paradigm. The size of the computational work varies from evaluating the interaction between two atoms to computing the contributions of one force for all atoms. Whenever a slave finishes its work, it notifies the master and gets new work. Table 1 gives an overview of the Apo A1 benchmark with 92224 atoms simulating 10 fs. We get clearly a better scaling for larger number of nodes since the work is distributed more evenly.

Nodes	Static [s]	Speedup	Dynamic [s]	Speedup
1	677.2	1	677.2	1
2	337.4	2.0	343.2	1.97
4	181.6	3.73	175.2	3.87
8	98.85	6.85	89.51	7.57
16	53.48	12.7	47.91	14.1
32	30.83	22.0	28.16	24.0

Table 1. *ApoA1 benchmark (92224 atoms, non-periodic) simulating 10 fs with a cutoff of 12 Å on an Origin2000 with 195MHz R10000 processors. The left column shows the scaling using static load balancing, and the right column shows the scaling using a master-slave load balance. For the latter implementation, there is a node that acts as both master and slave.*

4 Conclusions

We have obtained reasonable speedup (75% efficiency) using one-sided communication in 32 nodes of an Origin2000. Since we use one-sided communication and avoid global communication, we think our approach will scale in clusters as well. One-sided communication gives flexibility on how to manage communication, and makes our implementation simple and flexible. The design of PROTOMOL made it easy to implement different load balanc-

ing and decomposition strategies. The parallelization strategies already implemented are transparent to future application developers, and the generic design allows for better data and work distribution strategies.

For the future we will avoid replication, which will make our framework also suitable for system with much more than 10^6 atoms. For some forces we do not expect the same performance as for a spatial decomposition, but we have the possibility to rearrange the data to get a nearly spatial decomposition. For large scale parallel machines, the same overall design applies, but the details of the data managing change drastically. In particular, the load balancing strategy would use a hierarchy of masters to avoid bottlenecks. Also, one would need to use distributed data.

Acknowledgments

The calculations were performed at the Norwegian super-computing facilities in Bergen through a Norges Forskningsråd grant. One author was supported by NSF grant 1068136.

Bibliography

[1] D. M. Beazley and P. S. Lomdahl. Message-passing multi-cell molecular dynamics on the connection machine 5. *Parallel Computing*, 20:173–195, 1994.

[2] B. R. Brooks and M. Hodošček. Parallelization of CHARMm for MIMD machines. *CDA*, 7:16–22, Dec. 1992.

[3] D. Brown, H. Minoux, and B. Maigret. A domain decomposition parallel processing algorithm for molecular dynamics simulations of systems of arbitrary connectivity. *Computer Physics Communications*, 103:170–186, 1997.

[4] T. W. Clark, R. v. Hanxleden, J. A. McCammon, and L. R. Scott. Parallelizing molecular dynamics using spatial decomposition. In *Proceedings of the Scalable High-Performance Computing Conference*, pages 95–102, Los Alamitos, Calif., 1994. IEEE Computer Society Press.

[5] S. Haney and J. Crotinger. How templates enable high-performance scientific computing in C++. *Computing In Science & Engineering*, 1(4):66–72, Jul-Aug 1999. POOMA reference.

[6] Y.-S. Hwang, R. Das, J. H. Saltz, M. Hodošček, and B. R. Brooks. Parallelizing molecular dynamics programs for distributed-memory machines. *IEEE Computational Science & Engineering*, 2(2):18–29, Summer 1995.

[7] J. A. Izaguirre, J. Willcock, T. Matthey, T. B. Slabach, T. Steinbach, S. Stender, G. F. Viamontes, and J. Mohnke. Protomol: An object oriented framework for molecular dynamics. `http://www.cse.nd.edu/~lcls`, 2000.

[8] L. Kalé, R. Skeel, R. Brunner, M. Bhandarkar, A. Gursoy, N. Krawetz, J. Phillips, A. Shinozaki, K. Varadarajan, and K. Schulten. NAMD2: Greater scalability for parallel molecular dynamics. *J. Comput. Phys*, 151(1):283–312, May 1, 1999.

[9] T. Matthey and J. P. Hansen. Evaluation of MPI's one-sided communication mechanism for short-range molecular dynamics on the Origin2000. In *PARA2000 and Workshop on Applied Parallel Computing*, 2000.

[10] MPI-forum. MPI-2:Extensions to the Message-Passing Interface. http://www.mpi-forum.org/docs/docs.html.

[11] M. Nelson, W. Humphrey, A. Gursoy, A. Dalke, L. Kalé, R. D. Skeel, and K. Schulten. NAMD—a parallel, object-oriented molecular dynamics program. *Intl. J. Supercomput. Applics. High Performance Computing*, 10(4):251–268, Winter 1996.

[12] D. Okunbor and R. Murty. Parallel molecular dynamics using force decomposition. In P. Deuflhard, J. Hermans, B. Leimkuhler, A. Mark, S. Reich, and R. D. Skeel, editors, *Computational Molecular Dynamics: Challenges, Methods, Ideas*, volume 4 of *Lecture Notes in Computational Science and Engineering*, pages 483–494. Springer-Verlag, Nov. 1998.

[13] S. Plimpton and B. Hendrickson. A new parallel method for molecular dynamics simulation of macromolecular systems. *J. Comput. Chem.*, 17(3):326, 1996.

[14] J. G. Siek and A. Lumsdaine. The matrix template library: A unifying for numerical linear algebra. In *International Symposium on Computing in Object-Oriented Parallel*, 1998. Also available from `http://lsc.nd.edu/downloads/research/mtl/papers/mtl_poosc.pdf`.

[15] R. D. Skeel. Macromolecular dynamics on a shared-memory multiprocessor. *J. Comp. Chem.*, 12(2):175–179, January 1991.

[16] G. R. Smith and M. S. P. Sansom. Dynamic properties of Na^+ ions in models of ion channels: a molecular dynamics study. *Biophys. J.*, 75:2767–2782, 1998.

[17] T. Veldhuizen. Blitz++: The library that thinks it is a compiler. Conference presentation, Extreme! Computing Laboratory, Indiana University Computer Science Department, Sep 1998. http://oonumerics.org/blitz/blitztalk.ps.gz.

[18] J. Vincent and K. M. Merz. A highly portable parallel implementation of Amber using the message passing interface standard. *J. Comp. Chem.*, 11:1420–1427, 1995.

[19] A. Windemuth. Advanced algorithms for molecular dynamics simulation: The program PMD. In T. G. Mattson, editor, *Parallel Computing in Computational Chemistry*, Washington, D.C., 1995. ACS Books. In press.