Experiences with Asynchronous Parallel Molecular Dynamics Simulations

Marcus Dormanns, Walter Sprangers
RWTH Aachen, Lehrstuhl für Betriebssysteme
Kopernikusstr. 16, D-52056 Aachen, Germany
e-mail: contact@lfbs.rwth-aachen.de, phone: +49-241-807634

Abstract. *This article provides some insight in the runtime behavior of asynchronous methods for parallel molecular dynamics simulations that we recently introduced. From this, the advantages of the asynchronous mode of operation becomes very obvious, i.e. the abilities to hide message passing latencies, to reduce network congestion and to level out processing power and/or load fluctuations. The conclusions are validated by simulation runs on a network of up to 12 workstations, comparing the results to a synchronous mode of execution.*

Keywords: *molecular dynamics simulation, parallel processing, asynchronous algorithm, performance analysis.*

1 Introduction

Computer simulation of molecular dynamics is one of the major applications of high performance computing. Moreover it is an important assistance to material science and biotechnology, which have been identified as key-technologies for the future. The most time-consuming part of such simulations are force computations at the atomic level, which determine the exact particles' motion over time [1].

This paper deals with asynchronous parallel algorithms for the N-body dynamics problem (see [2] for various other examples). Asynchronous parallel algorithms are characterized in that they contain no blocking send/receive operations (at least up to a certain extend). Instead, they exchange messages with non-blocking send and non-blocking or redirecting receive operations and are able to deal with somewhat outdated information. Therefore, this class of algorithms possesses capabilities to

- hide even large communication latencies which are caused by e.g. inexpensive mainstream hardware (e.g. a network of workstations) or a bad ratio of computation to communication, frequently present in daily medium-size problems, and
- compensate short-term load and/or processing power fluctuations of the compute nodes (e.g. caused by background computation load on a parallel machine with time-shared multiprogramming).

However, these highly desirable properties are not free of charge. Instead, it has to be paid for them with a decay in calculation precision. Fortunately, this error can be bounded to above and is not unique to the asynchronous mode of operation, but also present in other strategies, frequently employed to speed-up computation (e.g. multipole algorithms [6]).

In distinction to other efforts in parallelizing molecular dynamics simulations (e.g.

[9]), which aim at developing the best suited parallelization of a given (classical) algorithm, our work is concerned with the roots of the problem. I.e. to relax the coupling between different subproblems, and even more specific, with the time-related issue.

In section 2, we give a brief outline of the computation methodology and summarize some analytical results regarding the precision decay. The following section 3 gives some insight into the mode of operation and the potential performance gain. This paper ends with section 4 by drawing some concluding remarks.

2 Outline of the Algorithm

Particle interactions are described by the potential, in which a specific particle i, located at $x_i(t)$, resides at time t caused by all the other particles:

$$(1) \qquad \Phi(x_i(t)) = \sum_{j=1,\, j\neq i}^{N} \varphi(x_i(t), x_j(t))$$

Typically, each pair potential $\varphi(x_i(t), x_j(t))$ is approximated by a polynomial in the distance of the particles. The dynamic is given by the resulting force acting on the particle, from which Newton's equations of motion determine the actual trajectory, which is numerically integrated in a time-step fashion. For short-range interactions, only those particles contribute to the potential which reside inside a given cut-off radius.

The asynchronous mode of operation is enabled by allowing position information to be incorporated in potential/force evaluations which are somewhat outdated:

$$(2) \qquad \Phi^{async}(x_i(t)) = \sum_{j=1,\, j\neq i}^{N} \varphi(x_i(t), x_j(t')) \text{ where } t' \leq t$$

It was shown in [4] that for a simulation domain, parqueted into small cells of size d (a fraction of the cut-off radius), a time step size Δt and an upper bound of the particles' velocities v_b, the maximum relative error in the potential can be bounded to above by

$$(3) \qquad \varepsilon_\Phi = 1 - \left(1 - \frac{v_b \cdot \Delta t}{d}\right)^p$$

where it is assumed that the degree of obsolescence in the particle positions (as a multiple of Δt) is at most proportional to their distance, measured in d. p is the exponent of a potential, stated as a one-term polynomial in the particles' distance. Furthermore, it was demonstrated, that this magnitude of the potential error stands in a good relationship to those encountered by other strategies, e.g. multipole algorithms [6]. Actual errors in a real simulation are much smaller, because the upper bound (3) relies on worst-case assumptions, e.g. regarding the velocity of the particles that determines the maximum displacement between t' and t. Furthermore, it was assumed that the directions (in physical space) of all displacements due to asynchronism which influence the error, are the same.

In [5], it was shown how to derive a parallel program from this abstract computation

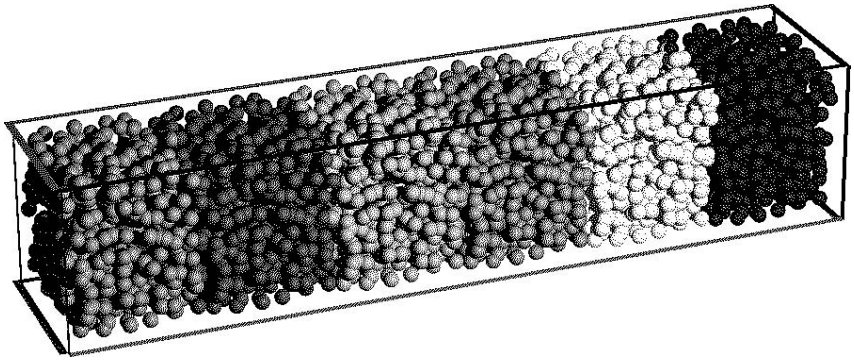

Figure 1: Exemplary simulation domain, parqueted with 10 x 10 x 48 cells, with 4800 particles. The domain is partitioned into 6 subdomains for parallel processing, with the particles shaded accordingly.

methodology exploiting the possibility of a process to employ outdated non-local data at times, using spatial decomposition of the simulation domain. This covers issues regarding a suitable (flexible) computation progress strategy and ensuring consistency of the data which is prone to message passing. The asynchronous mode of execution only makes sense, if also particle migration to maintain the spatial decomposition is performed asynchronously. To ensure that no particle gets lost or exists twice for a time step in one process, a suitable protocol has to be obeyed. Figure 1 exemplarily shows a decomposition of the simulation domain, on which the parallelization is based.

This methodology is strongly related to the so-called *multiple time step method*, proposed in [10], which evaluates forces with different frequency according to the distance. But in contrast to our method, the multiple time step method aims at reducing the (sequential) computational complexity, not the parallelization capabilities of the algorithm.

3 Communication Characteristics and Performance

To give a somewhat more detailed insight into the mode of operation, we discuss the progress of execution for the simple example of figure 1. Particles interact via the frequently employed short-range Lennard-Jones potential:

$$(4) \qquad \varphi(x_i(t), x_j(t)) = 4\varepsilon\left[\left(\frac{\sigma}{\|x_i(t) - x_j(t)\|}\right)^{12} - \left(\frac{\sigma}{\|x_i(t) - x_j(t)\|}\right)^{6}\right]$$

All simulations were performed on SUN SparcStation Classic, connected via standard Ethernet. To visualize the program behavior over time, the *upshot* utility was used, which comes with the MPICH MPI implementation [7] and provides similar capabilities to *ParaGraph* [8].

Figure 2 shows the different phases of the individual processes in the course of time

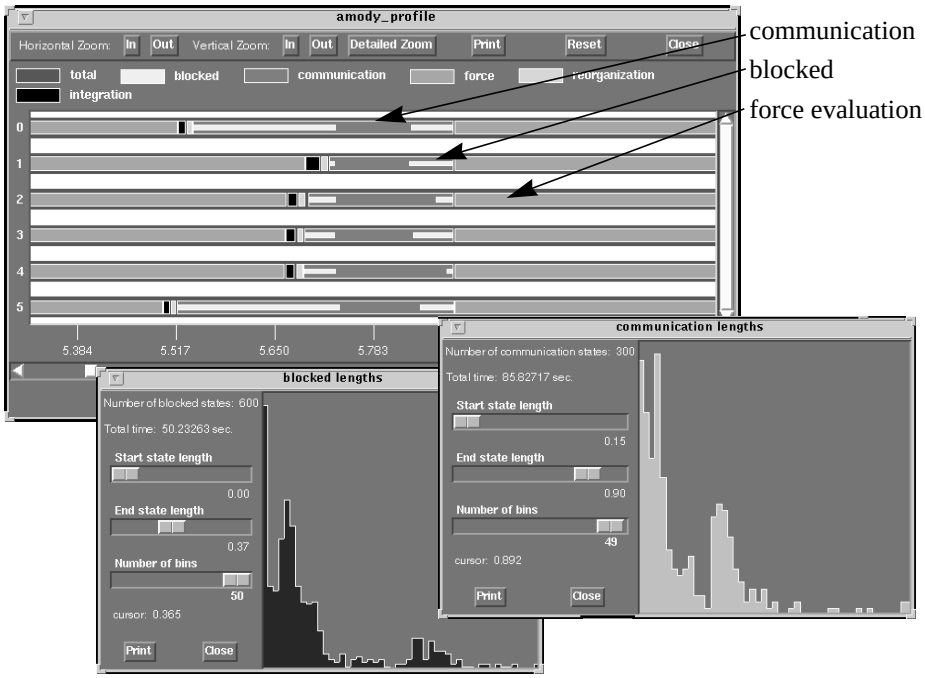

Figure 2: States of the individual processes in the course of time in a strictly synchronous execution mode. The total time and it's distribution among the occurrences is shown for the states *communication* and *blocked* (x-axis: duration, y-axis: frequency).

for a strictly synchronous mode of execution. From this, the root of the performance degeneration is quite obvious. A communication phase starts not before all processes arrive at this point.

This is due to the 3-way flow protocol, implemented in the MPI message passing library (see e.g. [3] for a discussion). Therefore, part of the communication time (total 85 sec.) consists of time, processes are just blocked (50 sec. light grey, surrounded by dark grey, indicating the whole communication phase). This synchronization forces all processes to send/receive data simultaneously, what makes the actual data exchange even more inefficient due to a lot of contention on the Ethernet.

In contrast to this, figure 3 shows a clip of the program behavior over time for an asynchronous framework. Therein, a process sends a message whenever it has collected all necessary data. So it happens that a process which has actually two communication partners sends its update messages to both at two different points in time. Messages are incorporated either, when a message has already arrived and the process is in a suitable state, or it exhausted all its progress credit due to asynchronism (remember the assumptions made in section 2 regarding the maximum degree of obsolescence allowed) and has to wait for a message anyway. However, within the employed simulation settings such a situation never occurred. So it is possible that a process incorpo-

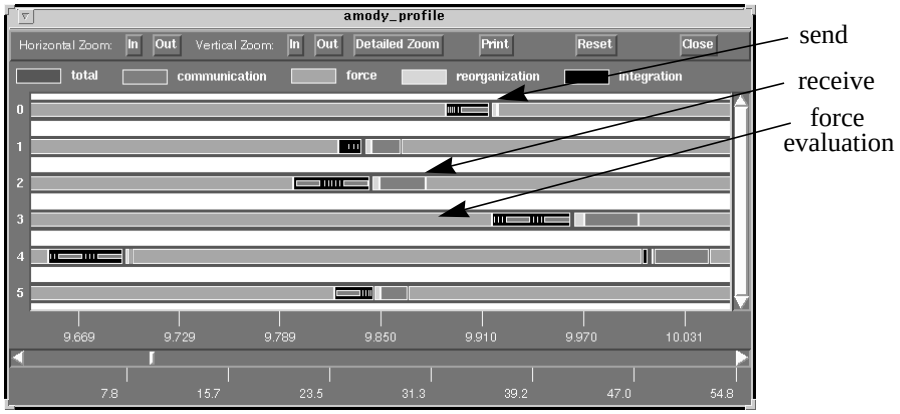

Figure 3: Course of time in an asynchronous mode of operation.

rates a message without sending an update messages itself (process 4 in figure 3). Eventually, it finds itself in a situation when it has computed all the necessary data to compile the next update message for one of its neighbors. Altogether, the communication overhead in the asynchronous framework for the simulated problem is just 18 sec., what is just 21% of that in the synchronous mode of operation.

While it could be recognized that the network bandwidth became a limiting factor in synchronous execution for a high degree of parallelism, it was not in an asynchronous mode of execution where the total amount of data to be exchanged is much more evenly distributed over time, resulting in less network congestion. Table 1 shows the obtained speed-up figures, which are significantly better for the asynchronous mode of execution.

# procs	speedup (sync.)	speedup (async.)
1	1.0	1.0
2	1.6	1.9
3	2.0	2.5
4	2.4	3.1
6	3.0	4.3
8	3.4	4.9
12	3.7	6.4

Table 1: Obtained speedup figures for up to 12 workstations within a synchronous and an asynchronous mode of execution.

Although not considered in the simulations presented in this paper, another point is worth mentioning. As is quite obvious, a parallel program executed in an asynchronous mode of operation possesses the principal capabilities to hide blocking times of individual processes due to short-term processing power or load fluctuations of other

nodes. This works best, if they are only evenly distributed among all participating processing nodes, their occurrences do not scatter to much in time and their durations are at most of the order of the extra computation progress credit due to asynchronism.

4 Conclusions and Outlook

The analysis of the runtime behavior confirms the expectations we had when we started thinking about asynchronism in parallel molecular dynamics simulations. As demonstrated, asynchronism is a powerful means to enhance performance of daily relevant, medium-size problems on mainstream hardware, i.e. a network of workstations. Moreover, as a computational methodology it is the next step beyond classical latency hiding techniques, e.g. restructuring of the code [11], in that its capability to relax synchronization constraints is much larger.

As a next step towards applying it to real production codes, we will study the behavior of the proposed kind of algorithms under the influence of background load in more detail. This is an important point, because very typical in real-world computation environments, where computations are not performed on a dedicated parallel machine but on a collection of desktop machines, which should be primary at the disposal of their owners.

References

[1] Allen, M. P.; Tildesley, D. J.: *Computer Simulation of Liquids.* Oxford University Press, 1987.

[2] Bertsekas, D. P.; Tsitsiklis, J. N.: *Parallel and Distributed Computation: Numerical Methods.* Prentice Hall, 1989.

[3] Butenuth, R.: *Message Passing in Parallel Operating Systems and Applications.* Proc. High Performance Computing ‘96, New Orleans.

[4] Dormanns, M.; Sprangers, W.: *On the Precision of Asynchronous Parallel Molecular Dynamics Simulations.* Technical Report, Chair for Operating Systems, RWTH Aachen, Oct. 1995.

[5] Dormanns, M. Sprangers, W.; Bemmerl, T.: *Feasibility of Asynchronous Parallel Molecular Dynamics Simulations.* Proc. High Performance Computing ‘96, New Orleans.

[6] Elliot, W. D.: R*evisiting the Fast Multipole Algorithm Error Bounds.* Technical Report 94-008, Duke University, Dept. of Electrical Engineering, Durham, NC, Jan. 1995.

[7] Gropp, W.; Lusk, E.;Skjellum, A.: *Using MPI: Portable Parallel Programming with the Message-Passing Interface.* MIT Press, 1994.

[8] Günther, H.; Bemmerl, T.: *Programming Scalable HPC Systems: Tools and their Application.* Speedup Journal, Vol. 9, No. 1, pp. 68-71, 1995.

[9] Plimpton, S.: Fast Parallel Algorithms for Short-Range Molecular dynamics. J. of Comp. Phy., Vol. 117, pp. 1-19, March 1995.

[10] Street, W. B.; Tildesley, D. J.; Saville, G.: *Multiple time step methods in molecular dynamics.* Mol. Phys., Vol. 35, No. 3, pp. 639-648, 1978.

[11] Strumpen, V.; Casavant, T. T.: *Implementing Communication Latency Hiding in High-Latency Computer Networks.* Proc. HPCN ’95, Springer LNCS, pp. 86-93, 1995.