FEASIBILITY OF ASYNCHRONOUS PARALLEL MOLECULAR DYNAMICS SIMULATIONS

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

Keywords: molecular dynamics simulation, parallel processing, asynchronous algorithm, data coherence.

Abstract. This article proposes asynchronism as an alternative computation methodology in parallel molecular dynamics simulations. As already shown for areas like iterative solution of linear equations, asynchronism is able to hide message passing latencies and load or processing power fluctuations. Therefore, this innovative computation strategy, in combination with a flexible scheduling strategy of computation progress, provides the basis for good scalability even for moderate-size problems. Besides algorithmic issues, first promising simulation results are presented and discussed.

1 INTRODUCTION

Parallelization of algorithms from different areas, employing the methodology of asynchronism, e.g. from iterative solution of linear equations (Bertsekas and Tsitsiklis 1989), combinatorial optimization (Hong and McMillin 1995) and event driven simulation (Lin and Fishwick 1995) has received a considerable amount of attention in the last years. This paper introduces asynchronism in parallel molecular dynamics simulations. Asynchronous parallel algorithms are characterized by the property, that they contain no blocking receive operations (at least up to a certain extend). Therefore, such an algorithm has to cope with individual processes dealing temporary with outdated information about the computation progress of other processes without becoming incorrect. Or it must be able to compensate this with marginal extra computation. Due to these properties, asynchronous parallel algorithms exhibit highly desirable properties, like:

- the ability of hiding large network latencies caused by inexpensive mainstream hardware or a bad ratio of computation to communication, frequently present for relevant problems which are of medium-size (in opposite to the so-called *grand-challenges*, e.g. (Lomdahl *et al.* 1993); see also (Plimpton 1995))
- the ability to compensate temporary load unbalance in dynamically changing computation structures of a problem or computation power variations of different time scales of the allocated compute nodes; these may arise in real-world computation environments, e.g. due to additional background load if the parallel platform is operating in an overlapping space-shared mode (like the very popular networks of workstations as an extreme example)

These are the typical circumstances which are responsible for poor efficiency and limit scalability of classic synchronous parallel molecular dynamics simulations. Not surprisingly, most studies of parallel molecular dynamics on networks of workstations do not consider more than just four or six machines (Amundsen 1993; Boryczko *et al.* 1994; Bubak *et al.* 1994). The methodology of asynchronism provides the potential to overcome these limitations.

The proposed strategy is a generalization and adoption of the *multiple time step method* proposed in (Street *et al.* 1978). Therein, a division of the interaction range of the particles in a primary component, wherein the interactions are updated each time step, and in a secondary component, wherein interactions are only computed every few time steps, leads to a reduction in (sequential) computation. In (Nakano *et al.* 1993) this method was employed in a parallel molecular dynamics code. But in distinction to our work, besides reducing computation it is just exploited to reduce the communication volume in parallel processing. Govindan and Franklin (Govindan and Franklin 1994) applied a methodology named *speculative computation*, to N-body simulations. They employed estimations of particle positions derived by first-order Taylor approximation (similar to (Nakano *et al.* 1993)). But due to the non-spatial particle decomposition of the problem, which can lead to large errors employing outdated information, they have to check their results afterwards when the real position information is available. Upon exceeding a certain error threshold, a rollback has to be performed, reducing the possible performance gain due to asynchronism.

In (Dormanns and Sprangers 1995) we demonstrated how the general strategy of asynchronism can be applied in principal to molecular dynamics simulation and proved it's justification by deriving reasonable error bounds for the precision decay in calculation. This effect is not unique to the method we propose, but common among other methodologies too (e.g. hierarchical multipole algorithms), which exchange computation complexity with precision. But in contrast to our approach, these strategies aim at reducing the sequential computation complexity, whereas we aim at increasing the potential for parallel processing. Besides the fact that the methodology obeys certain precision restrictions, a lot of implementation issues have to be considered.

The organization of the paper is as follows: in section 2, we give an outline of the algorithm and it's parallelization. In the following section 3, we identify issues and problems arising from asynchronism and discuss how they can be solved. We illustrate these solutions with some early simulation results in section 4 and finish with some conclusions in section 5.

2 OUTLINE OF THE ALGORITHM

Subject of this work are algorithms for molecular dynamics simulations and other N-body dynamics problems, which rely on a calculation of particle pair-interactions, defined by potentials of the general form

$$(2.1)\quad \Phi(x_i) = \sum_{j=1, j \neq i}^{N} \varphi(x_i, x_j)$$

Here, settings with short- or medium-range interactions are addressed, which are assumed to vanish beyond a certain radius, the cut-off radius r_c. Hence, interactions between particles with greater distance can be neglected, saving a lot of computation compared to the total of N(N-1)/2 pair-interactions.

The way, asynchronism comes into game is by allowing to incorporate somewhat outdated information in the potential calculation:

$$(2.2)\quad \Phi(x_i(t_1)) = \sum_{j=1, j \neq i}^{N} \varphi(x_i(t_1), x_j(t_2))$$

where t_2 may be smaller than t_1 within certain limits, caused e.g. by network latencies. For a detailed description and comparison to the multiple time step method, see (Dormanns and Sprangers 1995).

To be able to access all particles inside the cut-off radius of each individual particle efficiently without searching among all particles, the today frequently used strategy is to divide the simulation domain into cells with a fraction of the cut-off radius as their boarder lengths (see figure 2.1 top). The particles are assigned to the cells, complying their location in space. Therefore, all particles in the neighborhood of a given particle can be accessed by simply visiting the particle lists in the corresponding neighboring cells (Beazley and Lomdahl 1994). Because molecular dynamics simulations show a dynamic behavior, this assignment has to be maintained during the whole simulation run.

2.1 Parallelization in an Asynchronous Framework

The parallelization strategy we employ is based on a spatial decomposition of the simulation domain. Each

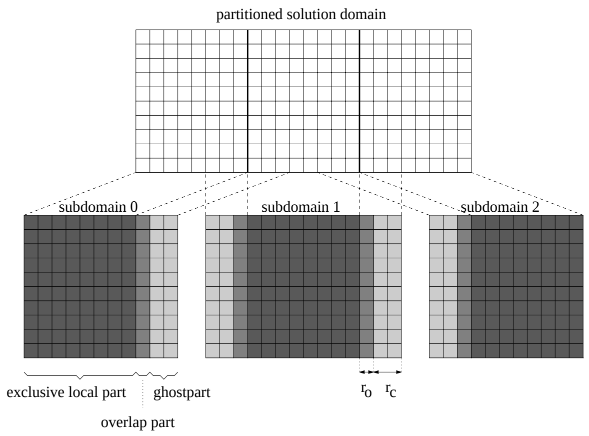


Figure 2.1: Top: 2D-simulation domain, parqueted with cells. Bottom: Partitioning into three subdomains with the enlargement due to an overlap domain and the ghost region.

processor is responsible for the calculation of all partial forces, acting on it's particles (Plimpton 1995). For may reasons, we deal with an one-dimensional decomposition of the simulation domain. This helps keeping the algorithm simple and reduces management overhead. Additionally, an one-dimensional decomposition simplifies load-balancing, which is highly desirable because of the dynamic properties of molecular dynamics simulations (Boryczko *et al.* 1994). Last but not least, the loss due to higher communication volume for larger domain boundaries for medium-grain parallelism remains small.

2.2 Data Organization

Meeting the requirements of the algorithm framework regarding partitioning into subdomains, communication and asynchronism issues (as will become clear later), the uppermost level of data structure organization is to divide the particles into *planes* of cells, perpendicular to the direction of partitioning.

To visualize the progress in computation, we depict the data of all particles in one plane of cells for one specific time step as a single box. According to the nature of the domain the box represents, it is shaded correspondingly. Besides the exclusive local data, data of neighboring processes in the range of the cut-off radius r_c is mirrored which is required to perform the local compu-

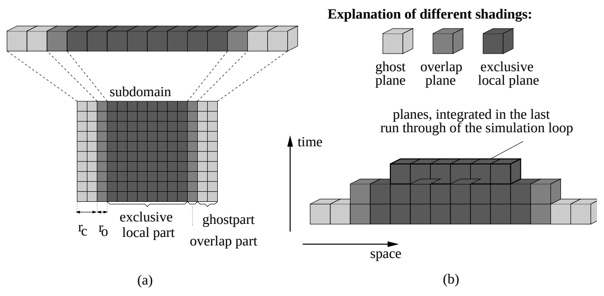


Figure 2.2: (a) Depicting planes of cells as boxes. (b) Visualization of computational progress in time.

tation. This can be further enlarged by an additional overlap domain of size r_o to enhance the possible degree of asynchronism, coming along with marginal redundant computation.

Boxes, i.e. planes of cells of the same time step are arranged in horizontal direction, while the progress in time is expressed along the vertical axis (see figure 2.2).

2.3 The Algorithm

Within each process, one sweep through the simulation loop of this lock-step algorithm starts with the determination of a subset of planes that is to be integrated to the next time step. In opposite to a synchronous mode of operation, where simply each plane is considered in each sweep, taking a snapshot in the asynchronous mode of operation, different planes can be integrated up to different time steps. A plane can be scheduled for integration if only the bounds are observed, which determine how many time steps the data of another plane inside it's cut-off radius is allowed to be outdated.

The `send` procedure of traditional molecular dynamics algorithms is adapted in that the send operation might not be supplied with all the necessary data of one time step and all the part of a subdomain necessary to compile a complete update message at once. Rather, the send operation has to collect newly integrated particle positions of the required planes from several sweeps to compile an update message for a neighboring process. Therefore, it must be able to store data for more than one update message in preparation at the same time. Due to the asynchronous mode of operation, the time step of some messages might be ahead the local integration process. Therefore, the incorporation of received data must be handled with care.

Particles moving across domain boundaries need to be migrated from the responsibility of one process to the neighboring one, to maintain the proper assignment of particles to cells not only inside one subdomain but also across process boundaries.

```
While ∃ plane ∈ {exclusive local domain}
      with plane.timestep < $t_{max}$ Do
   determine_integration_region(region);
   calculate_forces(region);
   integrate(region);
   send_update_messages(region);
   receive_update_messages();
   assignment_maintenance_interior(region);
   assignment_maintenance_boundary();
End While
```

Algorithm 2.1: Algorithmic skeleton of parallel multi-cell molecular dynamics simulation codes.

Algorithm 2.1 shows the general algorithmic skeleton as just explained.

3 PECULIARITIES DUE TO ASYNCHRONISM

Two main problem areas have been identified during this work. The first one concerns the strategy, which determines the scheduling of planes for integration, the second one the maintenance of data coherence in an asynchronous environment with dynamically changing problem structure. I.e. it must be guaranteed, that at any time the local view of each process sees each particle residing really in it's subdomain once and only once.

3.1 Computation Progress Strategy

In (Dormanns and Sprangers 1995) we derived a reasonable error bound for polynomial-like potentials in view of the sufficient condition that the differences in the time steps of two interacting particles is at most equal to their distance, measured in cells, minus one. Firstly, the choice of a suitable computational progress strategy is driven by this hard constraint. For sake of simplicity, we pick out an individual process $p \in \{0, ..., P-1\}$ and let it's planes be numbered from 0 to n_p-1. We can state this hard constraint for a plane i formally as a boolean predicate:

$$(3.1)\quad \begin{aligned} &\text{ready}\,(i) \equiv \forall j,\ 0 \le j \le n_p,\ i \ne j: \\ &(|\text{plane}\,[i]\,.\text{timestep} - \text{plane}\,[j]\,.\text{timestep}| \le |i - j| - 1) \\ &\quad \vee\ (j < i \wedge p = 0)\ \vee\ (j > i \wedge p = P - 1) \end{aligned}$$

where the last two conditions consider the cases where the subdomain of the process is just the left- or rightmost one.

The derivation of the computational progress strategy (which is interchangeable with the shape of the so-called computation progress skyline in the time-domain; see figure 2.2) is guided by the following considerations:

(1) the computation progress skyline should be as flat as possible, maximizing calculation precision,

(2) the computation progress of each process skyline should not contain any ‘valleys’ (so it should be unimodal); there is no reason for them, because if the planes at their ‘slopes’ were ready for integration, planes inside are also; this also maximizes calculation precision,

(3) progress in the overlap region of the subdomain should only be performed if nothing else can be done, avoiding redundant work whenever possible,

(4) the set of planes, scheduled for integration in a single sweep of the simulation loop should be as large and compact as possible to exploit Newton’s third law as much and as simple as possible, reducing computation complexity and

(5) the total set to be integrated in one sweep should be at least subdivided into a *left* and a *right* region, to meet the different progress situations at both domain boundaries caused by asynchronism.

The strategy we employ, obeys (2) and (5) as are and tries to incorporate the other points as far as possible.

At this point of the discussion, the justification for the additional overlap at the domain boundaries and the advantages of the asynchronous mode of operation becomes clear.

If only the required mirrored data (regarding the time step) of a neighboring process is available, no redundant work is spent in the overlap region. But if this data is late, e.g. due to computation power or load fluctuations within the processes, the flexible computation progress strategy enables an individual process to proceed with planes of cells inside the domain until the maximum progress is exhausted. The additional overlap domain enhances the amount of computation a single process can temporary be ahead of it’s collaborating processes. Therefore, load or computation power fluctuations of short time-scale can be leveled out without any efficiency decay.

Up to this point no asynchronism at all is required. But if messages are not just once but frequently late, e.g. due to network latencies, the total amount of progress that a process can be ahead will be exhausted soon. This is where asynchronism comes into game. Allowing processes to deal with somewhat outdated data enables the whole set of processes just to pass over this fact and proceed with computation, at least to the extend granted by the *ready* predicate.

3.2 Cell-Assignment Reorganization Inside one Process

This issue is raised by the observation, that particle migration between planes that are processed up to different time steps, is forbidden. The migration of a particle one time step behind the target plane means to consider the data of that particle (position, velocity, acceleration) from a preceding time step as the data of the time step of the target plane, which is indeed wrong. Therefore, it is not possible to reorganize the assignment of particles to cells regularly within the whole subdomain, as usually done in synchronous environments.

But for each time step and for each boundary between each two planes, there exists a sweep of the main simulation loop, when both are simultaneously at the same time step. These situations must be recognized and the assignment reorganization between these two planes performed at this time (this strategy is outlined in algorithm 3.1).

```
1  Function assignment_maintenance_local
           (region)
2    Foreach planeNo ∈ region Do
3      If plane[planeNo].timestep
         = plane[planeNo-1]
       and  planeNo,planeNo-1 not domain
           boundary
       and reassignment not already done
       Then
4        boundary_reorganize
               (planeNo-1,planeNo);
5      End If
6      If plane[planeNo].timestep
         = plane[planeNo+1]
       and planeNo,planeNo+1 not
           domain boundary
       and reassignment not already done
       Then
7        boundary_reorganize
               (planeNo,planeNo+1);
8      End If
9      inside_plane_reorganize(planeNo);
10   End Foreach
11 End Function
```

Algorithm 3.1: Maintenance of particle assignment to cells locally.

The boundary between exclusive local domains of different processes is not considered, it will be processed separately.

3.3 Coherence Protocols for Data Exchange

Due to the dynamic nature of the problem, i.e. particles moving around in the simulation domain, crossing cell and domain boundaries and the asynchronous mode of operation, some attention must be payed to ensure that no inconsistencies occur in the local view of the processes on non-local data.

3.3.1 Particle Migration across Process Boundaries

The first point under consideration is the migration of a particle residing in a cell of the exclusive local domain of one process to that of a neighboring process. In classical synchronous execution, the particle is migrated in one indivisible operation between two iteration steps, which never can lead to inconsistencies. On the other side, in an asynchronous mode of execution, it can happen with an unfavorable sequence of migration actions and update messages, that a particle temporary exists twice or gets lost at all. Such a situation can occur because the migration is not an indivisible operation, but asynchronously executed in both processes.

The protocol employed is shown in figure 3.1. The process, containing the cell in whose exclusive local subdomain a particle resides, is called the *owner* of that particle. Particle migration is always owner-initiated. It starts, when the owning process (i+1 in the figure) detects that a particle left it's exclusive subdomain, with sending a *migration-initiation* message to the corresponding neighboring process (marked with (1) in the figure). This process waits until itself reaches the time step at which the initiating process detected the boundary-crossing (2). Then, it performs the migration into it's local data structures and sends back an *acknowledge* message to the initiator. Upon reception (3), the initiator can also migrate the particle (4), being sure that in the meantime the target process became the owner and the particle cannot get lost. With this last step, the particle migration is completed (for performance reasons, a whole bulk of particles can be migrated in one operation). Temporary, the particle resides in both exclusive subdomains. Thus, a received update message of one or the other process (e.g. those depicted in the figure) may contain the particle of interest, while at the same time the particle is contained in it's exclusive local subdomain also. To avoid that the particle exists twice, incoming update messages are checked for those particles and erased from the update message.

This protocol rules out any problems which might occur if a particle moves back to the initiating process while the migration is still in progress, because a succeeding migration process cannot conflict with the first one.

3.3.2 Sending, Receiving and Incorporation of Particle Data

Due to the asynchronous mode of operation and the flexible scheduling of planes of cells for processing, two situations may arise forcing the

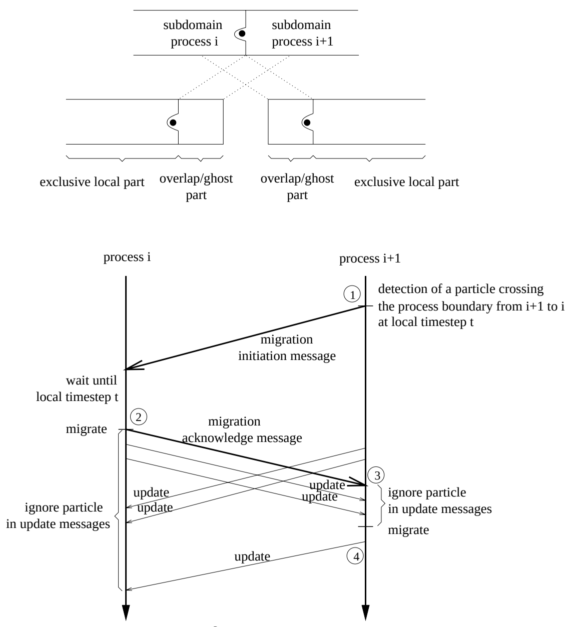
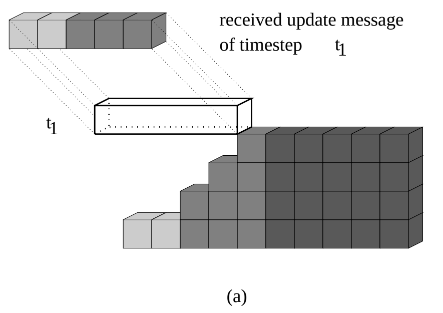

subdomain process i
subdomain process i+1
exclusive local part
overlap/ghost part
overlap/ghost part
exclusive local part
process i
process i+1
1
detection of a particle crossing the process boundary from i+1 to i at local timestep t
migration initiation message
wait until local timestep t
2
migrate
migration acknowledge message
3
update
update
update
update
ignore particle in update messages
ignore particle in update messages
migrate
update
4

Figure 3.1: Protocol for particle migration which ensures data consistency in an asynchronous environment.

```
Function assignment_maintenance_boundary()
  search for particles that left the
  exclusive local subdomain
  include them in a migration
  initiation message
  While an acknowledge message has
      been received
  or  a migration initiation message
      with timestep≥local timestep Do
    If acknowledge message Then
      perform migration too
    Else
      If timestep of initiation
        message not reached yet Then
        defer
      Else
        perform migration, send back
        acknowledge message
      End If
    End If
  End While
End Function
```

Algorithm 3.2: Particle to cell assignment maintenance at subdomain boundaries, implementing the proposed protocol.

`receive_update_message()` - function not to be able to incorporate the whole or part of the message. The first one is, when the sending process is in front of the target one (regarding the time step). In this case, the `receive_update_message()` - function simply defers it's incorporation until the local process reaches



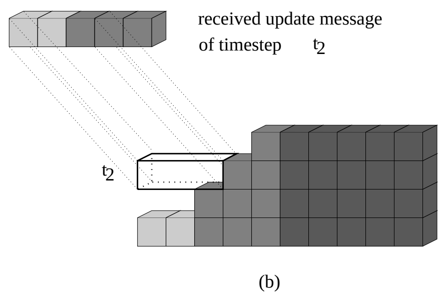


Figure 3.2: Two possible scenario, when an update message is received.

the same time step (figure 3.2a). Another possibility is that the local computation process has already overtaken the sending process at least with some of it's planes. In this situation, just the outermost part of the update message is incorporated, which does not overwrite already more actual data (figure 3.2b).

One issue remains to be solved: during the incorporation of two update messages, particles may have been reassigned to the cells, which could lead to particles existing twice or getting lost at the boundary up to that an update message was incorporated. To avoid this, a consistency check (and maybe re-establishment) has to be performed at this boundary.

The deletion of double particles must also be performed if an update message is completely incorporated, the loss of particles cannot occur in this case.

4 EARLY PERFORMANCE FIGURES

We report about some early test runs which should be sufficient to demonstrate the performance gain potential of the proposed asynchronous strategy in combination with the redundant overlap at subdomain boundaries. More detailed results are in preparation.

4.1 Simulation Setup

We consider the widely employed simulation of an ensemble of Argon atoms, whose interaction is described by the Lennard-Jones Potential

$$\varphi(x_i, x_j) = 4\varepsilon\left[\left(\frac{\sigma}{\|x_i - x_j\|^{12}}\right) - \left(\frac{\sigma}{\|x_i - x_j\|^{6}}\right)\right] \tag{4.1}$$

at reduced temperature T*=0.72 and reduced pressure ρ*=1.0 (see e.g. (Boryczko *et al.* 1994; Bubak *et al.* 1994; Lomdahl *et al.* 1993; Plimpton 1995)). The well-known Verlet integration scheme is used with a reduced time step size of t*=0.00462. The parameters ε and σ are taken from (Allen and Tildesley 1987). The simulation domain consists of $48 \times 10 \times 10$ cells of width σ, corresponding to 4800 particles. A cut-off radius of 3σ was employed. It should be noted, that the problem size is always the same, irrespective of the degree of parallelism. Boundary conditions are those of a massive wall, formed also by Argon atoms but with infinite mass. All tests were run on SUN SparcStations Classic, connected via standard Ethernet.

4.2 Results

One exemplarily snapshot of the particle positions during a simulation run is shown in figure 4.1. Figure 4.2 depicts the measured speedup for up to 12 workstations in absence of background load. As can be seen, the performance of the asynchronous version outperforms the synchronous one considerably. This is due to hiding message passing overhead and even more synchronization penalties, coming from different and varying computation complexity within the processes. Choosing r_o=1 leads to a more flat computation progress skyline, resulting in less computation due to a better exploitation of Newton's third law. For a large number of partitions, however, the computation overhead exceeds these savings.

As expected, the fraction of time spend in reorganization the assignment of particles to cells (locally as well as between different processes) was negligible.

5 CONCLUSIONS AND OUTLOOK

In this paper, we proposed a new methodology for parallel molecular dynamics simulations, especially suited for relevant, medium-size problems on mainstream hardware like networks of workstations, but not limited to. Although some peculiarities arise when implementing this strategy which do not exists in classical synchronous execution modes, it was shown how to solve them. The main differences affect the scheduling of the individual subproblems for computation, not the algorithmic skeleton itself, enabling this methodology to be employed also in parallelizing existing molecular dynamics simulation codes or enhancing already parallelized codes to meet the before mentioned environment actualities.

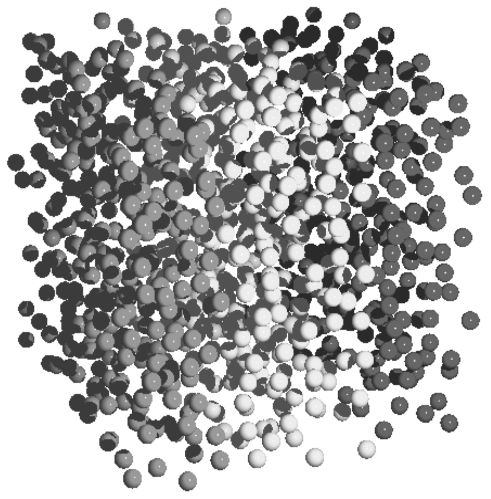


Figure 5.1: One simple illustrative snapshot during a similar simulation with 1000 particles employing three workstations. Particles are shaded corresponding to the workstation where they are processed on.

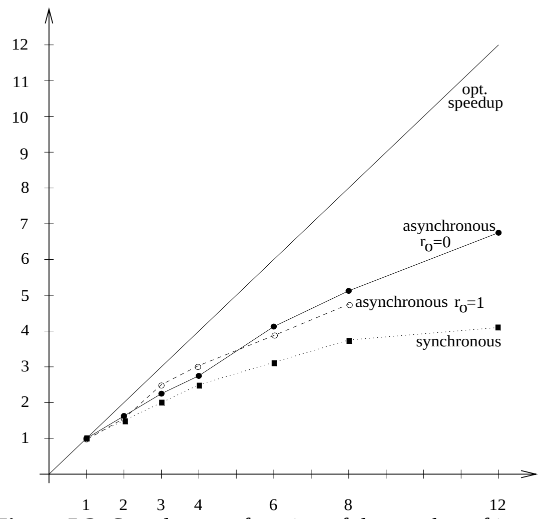


Figure 5.2: Speedup as a function of the number of incorporated workstations.

Although this paper demonstrated the methodology in principal, a lot of work still remains. Of primary interest is a more detailed evaluation of the performance enhancement potential, employing simulations as well as analytic considerations. The simulation results shown, already proved the justification of the proposed method. One interesting point is the behavior under additional background load. Another very important issue is the application of the concepts of asynchronism to real production codes, employed in the academic community as well as in industry. We intend to all these points in our future investigations.

ACKNOWLEDGEMENTS

We would like to thank Roger Butenuth from Univ. of Paderborn, Germany, for supplying his raytracer which was used to generate pictures from simulation runs.

REFERENCES

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

Amundsen, J.: *Distributed and Parallel Hamiltonian Molecular Dynamics.* Ph. D. thesis, Univ. Trontheim, Norway, 1993.

Beazley, D. M.; Lomdahl, P. S.: “Message-passing multi-cell molecular dynamics on the Connection Machine 5.” *Parallel Computing* 20, pp. 173-195, 1994.

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

Boryczko, K.; Kitowski, J.; Moscinski, J.: “Load-Balancing Procedure for Distributed short-range Molecular Dynamics.” In: Dongarra, J.; Wasniewski, J. (eds.): *Par. Sci. Comp., First Int. Workshop PARA ‘94*, LNCS 879, Springer, pp. 100-109, 1994.

Bubak, M.; Moscinski, J.; Pogoda, M.: “Parallel Distributed 2-D Short-Range Molecular Dynamics on Networked Workstations.” In: Dongarra, J.; Wasniewski, J. (eds.): *Par. Sci. Comp., First Int. Workshop PARA ‘94,* LNCS 879, Springer, pp. 127-135, 1994.

Dormanns, M.; Sprangers, W.: “On the Precision of Asynchronous Parallel Molecular Dynamics.” Tech. Report, Chair for Operating Systems, RWTH Aachen, Aug. 1995.

Durand, M. D.: “Cost Function Error in Asynchronous Parallel Simulated Annealing Algorithms.” Technical Report CUCS-423-89, Columbia University, June 1989

Govindan, V.; Franklin, M. A.: “Speculative Computation: Overcoming Communication Delays in Parallel Algorithms.” *Proc. Int. Conf. on Parallel Processing*, pp. III-12 - III-16, 1994.

Hendrickson, B.; Plimpton, S.: “Parallel Many-Body Simulations without all-to-all Communication.” Technical Report, Sandia National Labs and to appear in *J. Parallel and Distributed Computation.*

Hong, C.-H.; McMillin, B. M.: “Relaxing Synchronization in Distributed Simulated Annealing.” *IEEE Trans. on Par. and Distr. Systems*, Vol. 6, No. 2, Feb. 1995.

Lin, Y.-B.; Fishwick, P. A.: “Asynchronous Parallel Discrete Event Simulation.” Tech. Report, Univ. of Florida, *submitted to IEEE Trans. on Systems, Man and Cybernetics*, 1995.

Lomdahl, P. S.; Tamayo, P.; Gronbech-Jensen, N.; Beazley, D. M.: “50 GFlops Molecular Dynamics on the Connection Machine 5.” *Proc. Supercomputing ‘93*, pp.520-527.

Nakano, A.; Vashishta, P.; Kalia, R. K.: “Parallel multiple-time-step molecular dynamics with three-body interaction.” *Comp. Phy. Comm. 77*, pp. 303-312, 1993.

Plimpton, S.: “Fast Parallel Algorithms for Short-Range Molecular Dynamics.” *J. of Comp. Phy.*, Vol. 117, pp. 1-19, March 1995.

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.