Computer Physics Communications 124 (2000) 139–147

Computer Physics Communications

www.elsevier.nl/locate/cpc

Point-centered domain decomposition for parallel molecular dynamics simulation

R. Koradi [a], M. Billeter [b], P. Güntert [c,1]

[a] *Tripos, Inc., 1699 S. Hanley Road, St. Louis, MO 63144, USA*

[b] *Göteborg University, Biochemistry and Biophysics, Lundberg Laboratory, Box 462, S-40530 Göteborg, Sweden*

[c] *Institut für Molekularbiologie und Biophysik, Eidgenössische Technische Hochschule Hönggerberg, CH-8093 Zürich, Switzerland*

Received 15 April 1999; accepted 9 June 1999

Abstract

A new algorithm for molecular dynamics simulations of biological macromolecules on parallel computers, point-centered domain decomposition, is introduced. The molecular system is divided into clusters that are assigned to individual processors. Each cluster is characterized by a center point and comprises all atoms that are closer to its center point than to the center point of any other cluster. The point-centered domain decomposition algorithm is implemented in the new program OPALp using a standard message passing library, so that it runs on both shared memory and massively parallel distributed memory computers. Benchmarks show that the program makes efficient use of up to 100 and more processors for realistic systems of a protein in water comprising 10 000 to 20 000 atoms. © 2000 Elsevier Science B.V. All rights reserved.

Keywords: Molecular dynamics; Parallel computing; Message passing; Domain decomposition; MPI; OPALp

1. Introduction

Molecular dynamics (MD) simulations [1] have become a powerful tool for the investigation of macromolecular biological systems. They require large amounts of computation. Typical systems of a protein in water often consist of more than 10 000 atoms, and even if a cutoff for non-bonding interactions is used, several million interactions have to be calculated in each time step. Integration time steps cannot be made longer than about 2 fs to correctly sample high-frequency motions. Interesting simulations currently cover time ranges of 0.5–5 ns, and require on the order of 10^6 time steps. Longer simulations would be highly desirable, for instance to enable the simulation of protein folding processes. The new insight gained into the initial phase of protein folding from a recent, exceptionally long MD simulation of a protein with explicit representation of water for 1 μs [2] underlines the importance of efficient parallel algorithms for MD simulations.

Given these demands, it is clear that the most powerful computers are employed for performing the calculations. Traditionally these have been vector computers, which are very well suited for this problem. Recent tendencies in high performance computing show a trend away from specialized vector processors to parallel computers with many standard CPUs. Because the nature of the problem dictates that time steps can only be executed sequentially, the calculations for each time step need to be distributed, leading to fine-grained parallelism. Handling this distribution efficiently is

[1] E-mail: guentert@mol.biol.ethz.ch.

0010-4655/00/$ – see front matter © 2000 Elsevier Science B.V. All rights reserved.
PII: S0010-4655(99)00436-1

difficult, especially when trying to achieve high performance with large numbers of processors [3].

Two major strategies are commonly used for MD on parallel machines, e.g., [4,5]: The replicated data approach and the domain decomposition (or space decomposition) approach. In the replicated data approach, e.g., [6–8], all processing nodes hold the current values of all atom coordinates. The main advantage of this method is that implementation is relatively simple, it can often be done by local modifications to MD code that was originally written for serial execution. Also, because calculations can be assigned arbitrarily to any node, good load balancing is easy to achieve. However, because all coordinates need to be communicated to all nodes after each time step, the method becomes inefficient for large numbers of processors. In the domain decomposition approach, e.g., [2,4,9–12], the system is divided into regions in space. Each node is responsible for the calculations related to atoms in one of these regions. Since a cutoff is typically used for force calculations in large systems, each node only needs the coordinates of atoms from regions within the cutoff radius. This significantly reduces the communication requirements and enables much better scaling to large numbers of processors. Each node can also do additional calculations for atoms within its region, e.g., evaluating bonding potentials and performing the integration time step, such that also these calculations are distributed among the processors. Implementation is more complicated than in the replicated data approach, and good load balancing requires special attention.

OPALp uses a domain decomposition approach. Most published parallel MD simulation algorithms decompose the problem domain into equally sized rectangular boxes [10]. Srinivasan et al. [9] vary the sizes of the boxes to achieve better load balancing. We chose a different kind of space decomposition in which each region, in the following called "cluster", is defined by one point in space, called its "center". Each atom belongs to the region with the closest center. From a geometrical point of view, this decomposition resembles a three-dimensional Voronoi diagram [13]. This point-centered domain decomposition approach has a number of advantages over a decomposition into boxes:

(1) Different shapes of the problem domain cause no difficulties. A decomposition into boxes is natural if the problem domain is a box (or a trivial mapping of a box, like a rhomboid), but not favorable when, e.g., a truncated octahedron or an ellipsoid are used.
(2) Dynamic load balancing can be achieved through a straightforward modification of the definition of a cluster.
(3) Any number of clusters (processors) can be used easily. With boxes, the number of clusters is always the product of three integral numbers. Because it is normally beneficial for the boxes to have approximately equal sizes in all dimensions, this poses certain restrictions.
(4) The more "spherical" nature of the clusters may lead to a smaller number of "neighbors" (clusters within the cutoff radius). Whereas in a periodic system boxes always have at least 26 ($3 \times 3 \times 3 - 1$) neighbors, this number can be significantly lower with the present approach.

The point-centered domain decomposition method has been implemented in the new program OPALp which incorporates most of the functionality of its predecessor OPAL [14], with the additional option of applying a cutoff for non-bonding interactions and the possibility to use periodic boundary conditions. As in OPAL, the standard AMBER force field [15] is used without modification. The goal was to write a program that is highly efficient, considering both parallelization and vectorization, and easy to use. The well-structured and modular implementation makes OPALp a suitable platform for future development.

2. Parallelization strategy

2.1. Goals

The overwhelming part of all computation time in an MD simulation is spent for evaluating the potential of non-bonding interactions of atom pairs. A simple approach that is sometimes employed is to distribute only these calculations to multiple processors, collecting the resulting forces, and doing the rest of the calculation (bonding potentials, integration, etc.) on only one processor [7]. This works well with a small number of processors, but concentrating even a small part of the whole computation on only one processor will unavoidably create a bottleneck when going to many

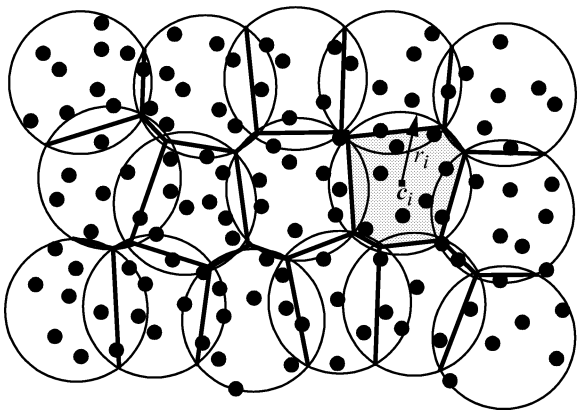

Fig. 1. Two-dimensional, schematic representation of the point-centered domain decomposition technique. Atoms are shown as black dots, the bounding spheres of clusters as circles, and the actual boundaries of clusters as lines. Cluster i, with center $\boldsymbol{c}_i$ and bounding sphere radius r_i (see text), is highlighted by shading.

processors [16]. Our approach distributes all parts of an MD simulation. Other points that have to be taken care of to enable efficient parallel execution are:

(1) Minimize the total amount of communication, where the number of communication messages turns out to be more critical than the communication volume.
(2) Wherever possible, overlap computation and communication, so that the time a processor waits for completion of communication requests is not wasted.
(3) Achieve good load balancing, so that each processor has approximately the same amount of computation to do.
(4) Maintain vectorization, because the single processors of many parallel machines are still vector processors.

2.2. Atom distribution

To make it possible that all parts of an MD simulation can be distributed to multiple processors, we chose a domain decomposition approach in which each atom is assigned to a processor, and this processor does all computations related to this atom, e.g., calculating the forces on the atom, integrating the equations of motion, doing velocity and coordinate scaling for temperature and pressure equilibration, etc. While it would be possible to randomly assign atoms to processors, it is possible to reduce the amount of communication by assigning groups of atoms that are close together in space to each processor. In the following, such a group of atoms is referred to as a "cluster". Each cluster is described by a point in space, called its "center". Atoms belong to the cluster whose center is closest to the atom (Fig. 1). The radius of a bounding sphere is calculated for each cluster by taking the maximum distance of its atoms from the center.

The initial distribution of atoms into clusters is obtained with an iterative algorithm. At first, the position of a randomly selected atom is taken as center for each cluster. All atoms are then assigned to the cluster with the closest center, and a new center for each cluster is calculated as center of gravity of the atoms assigned to it. This step is repeated 30 times, which consistently leads to a good start distribution. This algorithm would assign atoms to clusters based purely on their position in space, without taking chemical information into account. As will be explained later, it is beneficial to assign entire amino acid residues or solvent molecules to one cluster. To achieve this, the algorithm is slightly modified to assign all atoms of such a group to one cluster based on the closest distance of one of its atoms to a cluster center.

When using a cutoff for non-bonding interactions, the current coordinates of an atom are used only by processors that hold atoms within the cutoff distance. Because atoms are assigned to clusters based on their position in space, updated atom coordinates need only be communicated to a subset of all processors, as illustrated in Fig. 2. These clusters are called "neighbors". For most pairs of clusters, especially in large systems, the decision that they cannot be neighbors can be made by a quick calculation based on their bounding spheres. The gain of this approach becomes increasingly bigger when using large numbers of processors.

2.3. Pair list construction

When using a cutoff for non-bonding forces, it is necessary to construct a list of all pairs of atoms that are within the cutoff distance. This "pair list" is normally updated after a fixed number of MD steps, typically 10. For saving memory space, and for increased efficiency on vector computers, the list is not stored as

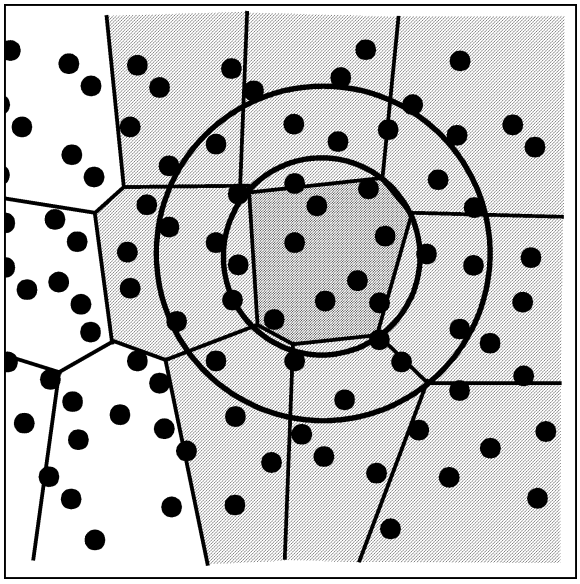

Fig. 2. A central cluster (shaded) and its neighboring clusters (hatched). The inner circle represents the bounding sphere of the central cluster. The radius of outer circle is given by the radius of the bounding sphere plus the cutoff distance for non-bonded interactions. The positions of atoms within the outer sphere are required by the processor that performs calculations for the central cluster.

a flat array of index pairs, but as a separate array of "partners" for each atom.

Since each processor is responsible for calculating non-bonding forces for the atoms that belong to its cluster, it only needs the part of the pair list where one of the atoms of the pair belongs to the cluster. Each processor calculates its own pair list. To do this, the first step is to get the current atom positions from all neighbors. Each processor can then build its pair list without further communication.

Because the force from one atom to another applies with opposite sign to the other atom, each atom pair must only be in the pair list of one processor, even if the two atoms are assigned to different processors. To minimize communication, as illustrated in more detail in the next section, it is beneficial if all forces between atoms from cluster i and atoms from cluster k are calculated by the same processor. Therefore, one of the two clusters is assigned to be the "master" of the pair. Whether cluster i is the master with respect to cluster k is defined by a pseudo-random rule:

$$\begin{aligned} &i \leqslant k : i \text{ master} \quad \text{if } k - i \text{ even}, \\ &i > k : i \text{ master} \quad \text{if } i - k \text{ odd}. \end{aligned} \tag{1}$$

In addition to the pair list, each processor creates, for each of its neighbors, a list of atoms that appear in the pair list and are assigned to this neighbor. This list gives the atoms for which coordinates and forces need to be communicated between MD steps. If cluster i is the master relative to the neighbor with index k, this list contains the indices of atoms belonging to k that i has interactions with. Cluster k also needs this list to determine which coordinates and forces it has to communicate to cluster i. To avoid duplicated calculations, these lists are communicated from the master to the slave of each cluster pair after the pair list is constructed.

To reduce artifacts, the cutoff is not applied to single atoms, but always to uncharged groups of atoms, i.e. protein residues or complete solvent molecules.

Bonded interactions (bond length, bond angle, dihedral angle, and improper dihedral angle potentials) are also assigned to processors when building the pair list. In the case of bond lengths, a bond between atoms that both belong to the same cluster is obviously handled by the processor assigned to this cluster. Bonds between atoms belonging to different clusters are handled by the processor that would also calculate a non-bonding interaction between the two atoms, i.e. the master of the two clusters. Even in the case of potential energy terms that involve more than two atoms, these atoms are never assigned to more than two different clusters because always entire residues are assigned to one cluster, and it is assumed that no bond angles or dihedrals involve atoms from more than two residues.

2.4. Molecular dynamics algorithm

As mentioned earlier, two key points of achieving high performance for a parallel program are to minimize the amount of communication, and to overlap communication and computation whenever possible. To overlap communication and computation, asynchronous communication is used, i.e. requests for obtaining data are issued ("posted") as early as possible, but are only required to have completed ("committed") later. The following pseudo code illustrates how this is done for an MD step. All steps are executed by each single processor:

(1) update pair list (if necessary)
(2) post receives of coordinates from slave neighbors
(3) send coordinates to master neighbors
(4) post receives of forces from master neighbors
(5) commit receives of coordinates
(6) calculate forces
(7) send forces to slave neighbors
(8) calculate kinetic energy
(9) commit receives of forces
(10) add forces received to forces locally calculated
(11) sum up kinetic energies and virials of all processors
(12) calculate new velocities, using kinetic energy for temperature control
(13) calculate new coordinates, using virial for pressure control

The communication demand is two messages for each pair of neighboring clusters in every MD step. One global communication step cannot be avoided if temperature and pressure control [17] are desired, e.g., the kinetic energy must be calculated for the complete system in order to apply temperature control. The communication library used (MPI) can accomplish the calculation of several sums with one communication primitive. In addition to the kinetic energy and virial which are mentioned in the pseudo code above, sums are also used for the center of mass (which is needed to calculate the virial and to neutralize the global motion of the system), and for a tensor that is used to neutralize global rotational motion of the system.

2.5. Dynamic load balancing

In Section 2.2, the initial assignment of atoms to processors was explained. For several reasons, this distribution can be far from perfect. For instance, when periodic boundary conditions are not used, there will be more non-bonding interactions for atoms in the middle of the system than for atoms near the boundary. Also if, due to the pseudo-random decision, a cluster happens to be master of significantly more than half of its neighbors, it has to do more calculations than other processors.

Even if a perfect a priori distribution could be found, this would not solve the load balancing problem. Because atoms move around during the MD simulation, the spatial relations change, and the distribution has to be adapted. Our algorithm does dynamic load balancing, and at the same time reassigns atoms to different clusters if they move away from one cluster.

A cluster i is defined by a point in space, $\boldsymbol{c}_i$, each atom is assigned to the cluster with the closest center, and the radius r_i of the bounding sphere is calculated as the maximum distance of an atom of the cluster to the center (Fig. 1). This definition is now extended to enable dynamic load balancing. We introduce a "bias" b_i, which is initially set to zero, but can later be changed to improve the load balancing. Each atom (with position vector $\boldsymbol{x}$) is then assigned to the cluster for which

$$|\boldsymbol{x} - \boldsymbol{c}_i|^2 - b_i = \text{minimal}. \tag{2}$$

When building the pair list, each processor stores the number n_i of non-bonding interactions it has to calculate. With perfect load balancing, all n_i's would be equal. The goal is now to approach this ideal state by modifying the b_i's. When b_i is increased, the cluster will grow, if it is decreased, the cluster will shrink. To estimate the necessary modification, the average of n_k over all neighbors k of cluster i is calculated:

$$a_i = \text{avg}(n_k) \quad k \in \text{neighbors}(i), \tag{3}$$

and the updated value of b_i' is obtained by

$$b_i' = b_i + \alpha r_i^2 \left(\left(\frac{a_i}{n_i} \right)^{2/3} - 1 \right). \tag{4}$$

This causes a relative volume change of the bounding sphere that is proportional to a_i / n_i. α is a constant that avoids making excessively large corrections. Empirical tests have shown that a value of $\alpha = 0.05$ leads to good load balancing within a few steps, while avoiding overcorrections.

Once the new bias is calculated, each processor tests whether the atoms currently assigned to it still belong to the same cluster, or whether they need to be moved to another cluster. An atom being moved to another cluster can be caused by the modified values of b_i', by the position of the atom being changed during the MD steps, or by a combination of both. The lists of atoms that move from one processor to another are communicated to all processors, and the data structures are updated accordingly.

Once the new distribution is established, each processor calculates the new center $\boldsymbol{c}_i$ of its cluster as center of gravity of all its atoms, and the new radius r_i'

as the maximum distance of one of the atoms from this center. The modification of the radius requires another adaption of the bias:

$$b_i'' = b_i' - \frac{r_i' - r_i}{2}. \quad (5)$$

This dynamic load balancing scheme can be applied before each update of the pair list, or less frequently if desired by the user.

As in Section 2.2, this description of the algorithm is slightly simplified. It does not take into account that groups of atoms (complete residues and solvent molecules) will always be assigned to one cluster. Again, this is achieved by a small modification that assigns all atoms of such a group to the cluster with the center closest to an atom in the group.

2.6. Energy minimization

Energy minimization using a local minimizer, e.g., the conjugate gradient method, is frequently used to remove strongly unfavorable interactions in the start structure of a molecular dynamics run. Most of the parallelization strategy described for molecular dynamics in the previous sections applies also to the implementation of energy minimization in OPALp. Calculation of the forces (steps 1 to 10 in Section 2.4) is identical. Once the forces are obtained, they are collected on a single CPU, which performs a step of minimization using the conjugate gradient method. The updated coordinates are then sent out to all CPUs with a broadcast operation. Unlike the algorithm for molecular dynamics, energy minimization performs a small part of the calculation for each step on a single CPU, and will therefore not scale equally well to large numbers of CPUs. However, since energy minimization is a quick operation that requires far fewer steps than an interesting molecular dynamics simulation, this is not a practical problem.

3. Implementation

The program OPALp was implemented using Fortran 90 [18], making use of its new features compared to Fortran 77. Functions and data are grouped into modules. Dynamic memory allocation is used for all data structures, there are no compile-time limits for the number of atoms, the size of pair lists, etc. The code was written to enable vectorization wherever possible.

MPI (Message Passing Interface) [19,20] was used for communication between tasks of the parallel program. The MPI standard has been widely accepted, is available for all important parallel machines, and makes it possible to write portable parallel programs that run efficiently on both shared memory and distributed memory machines.

The user interface of Opalp is based on the macro language INCLAN [21].

4. Results

4.1. Test system

The point-centered domain decomposition algorithm was tested in a free MD simulation of the protein cyclophilin A, starting from the structure that has been determined by NMR [22]. Out of the bundle of energy-minimized NMR conformers [22] the one with the lowest DYANA target function value [21] was immersed in a pre-equilibrated periodic box of TIP3P water [23] with a minimal distance of any protein atom from the boundary of 7 Å. This required 4735 water molecules. Together with the 2503 atoms of the protein, the entire system consisted of 16 708 atoms. 300 steps of energy minimization, as described in Section 2.6, were performed.

As described in the previous section, OPALp continually improves the load balancing, and only achieves good load balancing after the simulation runs for some time. This is not a problem in practical applications where long trajectories are generated, but the benchmark results, which should reflect the performance once good load balancing is achieved, should not be falsified by this effect. For this purpose, 50 MD steps of 1 fs were performed at the start of each calculation, with an update of the pair list and dynamic load balancing after each step. This was then followed by the part of the simulation that was timed for obtaining the benchmark results. Depending on the computer speed and number of processors, variable numbers of steps of 2 fs were performed in order to achieve runtimes between 10 and 30 minutes. In the following sections always the average wall-clock time per time step is given.

Table 1
Benchmark results on Cray J90

CPUs	Time (s)	Speedup	Efficiency (%)
1	5.65	1.00	100
2	3.21	1.76	88
3	2.12	2.67	89
4	1.68	3.36	84
5	1.39	4.07	81
6	1.17	4.84	80
7	1.08	5.22	75
8	1.05	5.38	67

A cutoff of 10 Å for non-bonded interactions was used in all benchmark calculations. Temperature and pressure equilibration [17] was applied, as well as periodic boundary conditions. The pressure was calculated from the virial [1,17], and the length of bonds involving hydrogen atoms was kept fixed using the SHAKE algorithm [24]. Overall translational and rotational motion of the protein was neutralized after each step.

4.2. Cray J90

The Cray J90 is a small vector computer with a peak performance of 200 MFLOPS per processor, supporting up to 32 processors with shared memory. A fully dedicated system with 8 processors was used for the benchmark. The pair list was updated after every 10 steps, load balancing was also done after every 10 steps. The performance of OPALp was 67 MFLOPS on one processor, indicating that good vectorization has been achieved. Table 1 shows execution times, speedups, and parallel efficiencies for 1–8 processors. Denoting with t_n the execution time with n processors, speedup is given by t_1/t_n, and parallel efficiency by $t_1/(nt_n)$. Using 2–6 processors, the parallel efficiency remains nearly constant and in the range of 80–89%. Dividing the system into clusters creates a certain overhead that is responsible for about 10% reduction in parallel efficiency (88% efficiency with two processors). On 8 processors (all processors of this machine) the parallel efficiency drops to 67%, an effect that can probably be attributed to the operating system performing some tasks even on a dedicated system. The absolute wall-clock times in the range of 1–2 s per MD step indicate that for a system of 16,000 atoms, an MD trajectory of 1 ns can be obtained in less than one week even on a small vector/parallel computer like the Cray J90.

4.3. Cray T3D

The Cray T3D is a massively parallel machine with (in our case 256) DEC Alpha CPUs and distributed memory. Benchmark results are well reproducible because the CPUs are fully dedicated to one task, without being interrupted by other jobs, interactive processes, or operating system tasks. Due to this feature and the large number of CPUs, we chose the T3D as the primary benchmark system, even though the program development was originally done on a Cray J90.

Table 2 shows the results when an update of the pair list as well as dynamic load balancing was performed every 10 steps. The results show that the program is able to make efficient use of a massively parallel system with distributed memory. With up to 64 processors, significantly more than half of the ideal parallel performance is achieved. The effectiveness of the dynamic load balancing algorithm is illustrated by a comparison with results obtained with less frequent load balancing, i.e. with an update of the pair list after every 10 steps but load balancing only after every 100 steps (Table 2). In general, speedup factors are significantly higher if dynamic load balancing is performed often (every 10 steps). Between 8 and 20% of the total computation time on 1–128 processors is due to updating the pair list and dynamic load balancing. With 256 processors, which corresponds to clusters of on the average only 65 atoms per processors, parallel performance drops to about 25%. This degradation is in part due to the fact that updating the pair list and load balancing becomes a bottleneck requiring 38% of the total execution time. In addition, it becomes difficult to achieve perfect load balancing if the cluster size becomes as small as 65 atoms because the clustering algorithm always keeps entire amino acid residues together in one cluster.

These results compare favorably with those reported for programs using either the data replication approach [6–8] or rectangular boxes for domain decomposition. Using rectangular domain decomposition Jabbarzadeh

Table 2
Benchmark results on Cray T3D

CPUs	Load balancing every 10 steps			Load balancing every 100 steps		
	Time (s)	Speedup	Efficiency (%)	Time (s)	Speedup	Efficiency (%)
1	23.9	1.00	100			
2	12.4	1.92	96			
4	6.42	3.73	93	6.34	3.77	95
8	3.29	7.27	91	3.40	7.04	88
16	2.10	11.4	71	2.31	10.3	65
32	1.22	19.6	61	1.79	13.4	42
64	0.663	36.1	56	0.817	29.3	46
128	0.476	50.2	39	0.517	46.3	36
256	0.411	58.3	23	0.378	63.3	25

et al. [12] obtained speedups of around 10–11 with 28 CPUs for systems of similar size, and they concluded that efficiency decreases significantly because of the increased amount of communication when using higher numbers of CPUs. Brown et al. [10] found comparable speedups for up to eight processors in simulations of a protein–water system of 21,423 atoms but no results were given for more than eight processors. OPALp achieves speedups of 11 already with 16 processors, and, more importantly, good efficiency is maintained up to at least 64 processors. In addition to its good scalability, the point-centered domain decomposition algorithm will save significant amounts of computation time in the simulation of solvated biological macromolecules because the shape of the simulation domain can be adapted to the shape of the protein. In this way the ratio between solvent and protein atoms is decreased by "cutting the corners" of an otherwise rectangular system without reduction of the thickness of the hydration shell around the macromolecule. Thus, we conclude that the point-centered domain decomposition algorithm is particularly advantageous for the simulation of complex biomolecular systems, whereas good parallel efficiency for homogeneous systems composed of identical small molecules can be achieved already by the simpler approach of rectangular domain decomposition, e.g., [5].

5. Conclusions

Our approach for MD calculations on parallel computers shows that, despite the difficulty inherent in fine-grained parallelization, it is possible to reach good efficiency also on massively parallel computers. The critical quantity that limits the speedup is the number of atoms divided by the number of CPUs. Our algorithm gives good speedups with as few as approximately 100 atoms per CPU on current architectures, and therefore can make good use of up to 100 and more processors for interesting problems because *all* calculation steps have been distributed to different processors. In contrast to domain decomposition methods based on box-shaped subdivisions of the system, our method poses no restrictions on the number of processors to be used. Furthermore, our parallel algorithm maintains good vectorization.

While high efficiency was a prime goal, there is still room for further possible optimizations that might lead to even higher performance of OPALp: Improved efficiency can be expected on particular machines by careful profiling and tuning. The MPI features for defining topologies could be used in order to allow the runtime system of the computer to make an ideal mapping of the problem to the physical layout of the machine (e.g., the processors of a Cray T3D are connected as a 3-dimensional torus). There are

possibilities for more overlap between calculation and communication if the forces within one cluster could be calculated before the receives of coordinates from other processors are completed. Finally, the dynamic load balancing algorithm could probably be refined further.

Acknowledgments

This work was supported with a grant from Cray Research to P. Güntert and K. Wüthrich and conducted as part of the ETHZ-Cray Cooperation project. We thank Carol Beaty of Cray Research for technical support. The benchmarks were done on a Cray J90 of the Eidgenössische Technische Hochschule Zürich (ETHZ) and on the Cray T3D of the Ecole Polytechnique Fédérale de Lausanne (EPFL), with help of the system administrators Bruno Loepfe (ETHZ) and Jean-Marc Chenais (EPFL). We thank Dr. Helmut Grubmüller for helpful discussions.

References

[1] M.P. Allen, D.J. Tildesley, Computer Simulation of Liquids (Oxford University Press, Oxford, 1987).
[2] Y. Duan, P.A. Kollman, Science 282 (1998) 740.
[3] M.F. Crowley, T.A. Darden, T.H. Cheatham III, D.W. Deerfield II, J. Supercomputing 11 (1997) 255.
[4] S. Plimpton, B. Hendrickson, J. Comp. Chem. 17 (1996) 469.
[5] M.R. Wilson, M.P. Allen, M.A. Warren, A. Sauron, W. Smith, J. Comp. Chem. 18 (1997) 478.
[6] S.E. Bolt, P.A. Kollman, J. Comp. Chem. 14 (1993) 312.
[7] J.J. Vincent, K.M. Merz, Jr., J. Comp. Chem. 16 (1995) 1420.
[8] F. Müller-Plathe, W. Scott, W.F. van Gunsteren, Comput. Phys. Commun. 84 (1994) 102.
[9] S.G. Srinivasan, I. Ashok, H. Jonsson, G. Kalonji, J. Zahorjan, Comput. Phys. Commun. 102 (1997) 44.
[10] D. Brown, H. Minoux, B. Maigret, Comput. Phys. Commun. 103 (1997) 170.
[11] K. Lim, S. Burnett, M. Iotov, R.B. McClurg, N. Vaidehi, S. Dasgupta, S. Taylor, W.A. Goddard III, J. Comp. Chem. 18 (1997) 501.
[12] A. Jabbarzadeh, J.D. Atkinson, R.I. Tanner, Comput. Phys. Commun. 107 (1997) 123.
[13] G. Voronoï, J. Reine Angew. Math. 134 (1908) 198.
[14] P. Luginbühl, P. Güntert, M. Billeter, K. Wüthrich, J. Biomol. NMR 8 (1996) 136.
[15] W.D. Cornell, P. Cieplak, C.I. Bayly, I.R. Gould, K.M. Merz Jr., D.M. Ferguson, D.C. Spellmeyer, T. Fox, J.W. Caldwell, P.A. Kollman, J. Amer. Chem. Soc. 117 (1995) 5179.
[16] G. Amdahl, Proc. Amer. Fed. Information Processing Soc. 30 (1967) 483.
[17] H.J.C. Berendsen, J.P.M. Postma, W.F. van Gunsteren, A. DiNola, J.R. Haak, J. Chem. Phys. 81 (1984) 3684.
[18] M. Metcalf, J. Reid, Fortran 90 Explained (Oxford University Press, Oxford, 1990).
[19] Message Passing Interface Forum, Int. J. Supercomputing Appl. 8 (1994) 1.
[20] W. Gropp, E. Lusk, A. Skjellum, Using MPI. Portable Parallel Programming with the Message Passing Interface (MIT Press, Cambridge, MA, 1994).
[21] P. Güntert, C. Mumenthaler, K. Wüthrich, J. Mol. Biol. 273 (1997) 283.
[22] M. Ottiger, O. Zerbe, P. Güntert, K. Wüthrich, J. Mol. Biol. 272 (1997) 64.
[23] W.J. Jorgensen, J. Chandrasekhar, J.D. Madura, R.W. Impey, M.L. Klein, J. Chem. Phys. 79 (1983) 926.
[24] J.-P. Ryckaert, G. Ciccotti, H.J.C. Berendsen, J. Comput. Phys. 23 (1977) 327.