Parallelization using Spatial Decomposition for Molecular Dynamics

Terry W. Clark
Reinhard v. Hanxleden
J. Andrew McCammon
L. Ridgway Scott

CRPC-TR93356-S
November 1993

Center for Research on Parallel Computing
Rice University
P.O. Box 1892
Houston, TX 77251-1892

Parallelization using Spatial Decomposition for Molecular Dynamics

Terry W. Clark
clark@kacha.chem.uh.edu
Dept. of Computer Science
University of Houston
Houston, TX 77204

Reinhard v. Hanxleden
reinhard@rice.edu
Dept. of Computer Science
Rice University
Houston, TX 77251

J. Andrew McCammon
mccammon@uh.edu
Dept. of Chemistry
University of Houston
Houston, TX 77204

L. Ridgway Scott
scott@uh.edu
Dept. of Mathematics
University of Houston
Houston, TX 77204

1 Introduction

Molecular dynamics (MD) simulations are useful computational approaches for studying various kinetic, thermodynamic, mechanistic, and structural properties [11, 14]. Molecular dynamics programs tend to be complex, taking many years to write, with frequent modification. There exist several MD programs, like GROMOS [7] or CHARMM [1], that are well established and are routinely used to solve a broad range of different simulation problems. However, despite the maturity of these programs and the significant hardware improvements made since their introduction, there is an interest in simulating larger systems, over longer periods of time, than is currently feasible. A number of researchers have already shown that MD is amenable for parallelization [2, 10, 6, 3, 19, 12]. However, certain difficulties arise when trying to achieve high efficiencies with large numbers of processors, largely due to the computationally irregular nature of MD codes in general. This paper presents EULERGROMOS, a parallelization of GROMOS that focuses on overcoming these scalability problems.

An MD simulation applies Newton's equations of motion to a molecular system to determine a new set of positions and velocities for the atoms at each timestep. The calculation of a single timestep involves an iteration over several major phases in the MD program. EULERGROMOS parallelized all of those major phases, including numerical integration with constraints (or SHAKE [20]), pairlist construction, and the computation of non-bonded forces (NBF). Since most MD runs perform the bulk of the work (around 90%) in the NBF routine, this phase is of particular interest. A common technique to accelerate the NBF calculation is to ignore all NBF interactions beyond a certain *cutoff radius*, R_{cut} [18]. This in turn provides an access locality that makes Eulerian, application space oriented data distributions desirable. An Eulerian decomposition, or geometric decomposition, assigns application space to processors, as compared to Lagrangian mappings where particles are assigned to processors.

A typical limitation of cutoff-radius based link-cell parallelizations is to restrict communication to nearest neighbors; consequently, each processor subdomain has to be greater than or equal to the cutoff radius size (see Section 4). Under this restriction, typical values for the density ρ of 0.1 $atoms/\text{Å}^3$ and R_{cut} of 15Å would result in a minimum of about 300 atoms per processor subdomain. For a fixed problem size, this restriction imposes an upper bound on the number of processors that can be used. However, as shown in Figure 1, EULERGROMOS can make effective use of more processors than would be allowed under this restriction. EULERGROMOS, with the GROMOS molecular dynamics program for its MD kernel, uses a spatial decomposition, but does not limit communication to nearest neighbors. Each processor's overlap region (of thickness R_{cut}) is allowed to penetrate several processor subdomain layers. Figure 1 shows that substantial performance gains would not be available otherwise.

EULERGROMOS provides a choice between two different communication algorithms, one of which (based on shifts) becomes faster as the number of processors increases. Efficient communi-

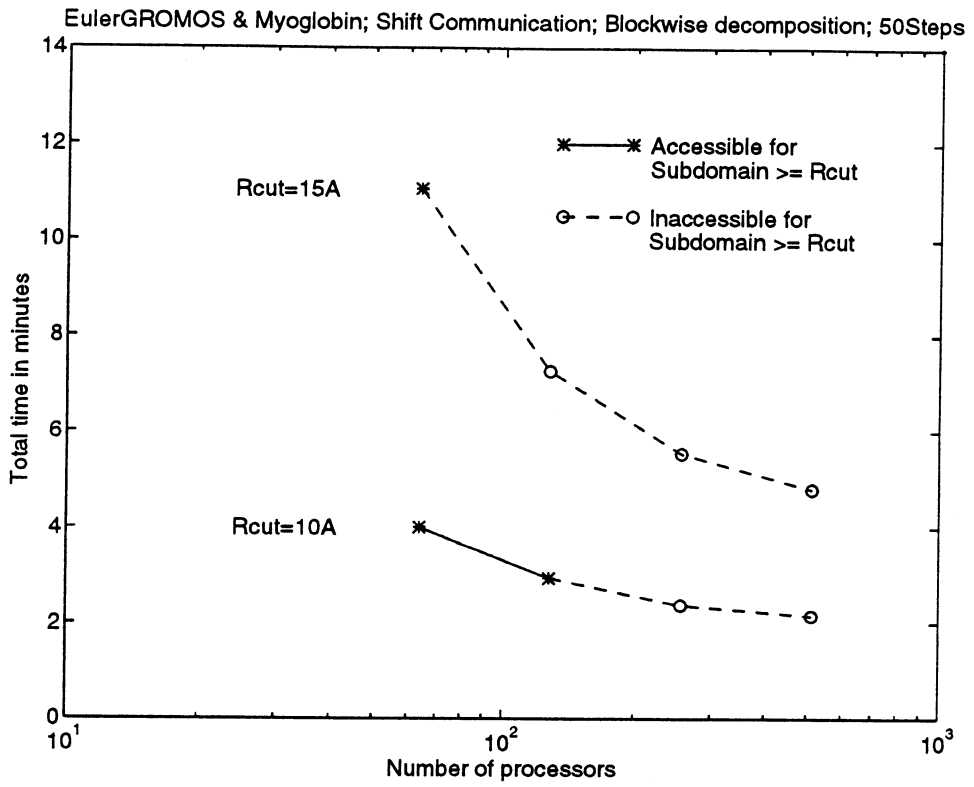


Figure 1: Processor accessibility limitation for the standard parallel link-cell method due to subdomain size.

cation, together with low parallelization and distribution overheads, allowed speedup gains with as few as approximately 20 atoms to a processor, including all phases of the molecular dynamics with constraints and I/O. Another characteristic of EULERGROMOS is the use of dynamic load balancing, which adjusts individual subdomain sizes to accommodate inhomogeneous atom densities.

The implementation of EULERGROMOS is described in more detail in Section 2. Section 3 evaluates the characteristics of the resulting program, presenting performance results for two biomolecular systems. One of these systems, the solvated acetylcholinesterase dimer (AChE) shown in Figure 2, contains over 100,000 atoms, about an order of magnitude more than typically simulated. Section 4 discusses related work, Section 5 concludes with a summary.

2 EulerGromos

2.1 Imposing a granularity on the problem domain

Using an Eulerian decomposition for distributing atoms and the computations associated with them improves inter-processor locality, which in turn increases scalability and reduces communication costs. However, to make good use of the intra-processor memory hierarchy as well (*i.e.*, to reduce cache misses), increasing access locality is also desirable within a processor. We therefore conceptually divide our overall ***problem domain***, which here is the physical space occupied by the set of atoms we want to simulate, into small rectilinear regions of fixed size, henceforth called ***subboxes***. There are n_d subboxes along dimension d, resulting in a total on $n_1 * n_2 * n_3$ subboxes. Each subbox contains a list representation of the atoms resident within its spatial extent [4]. To

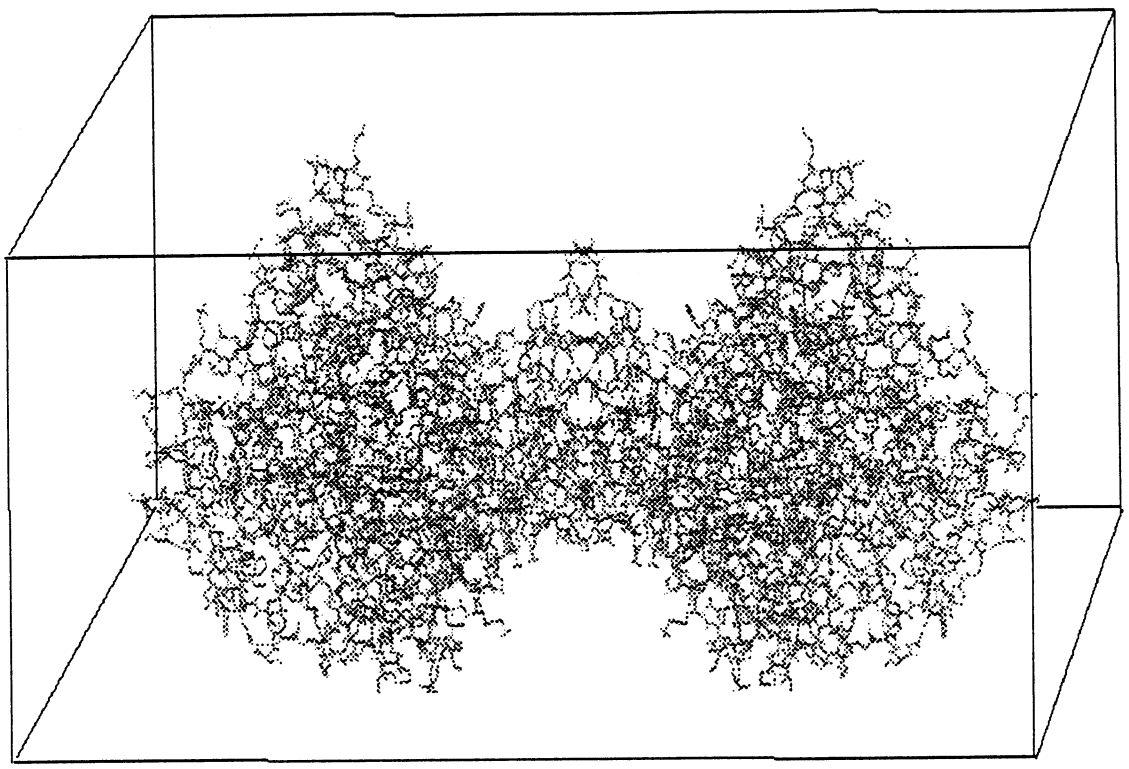


Figure 2: The 10,406 atom acetylcholinesterase dimer enclosed in a $91 \times 97 \times 106\text{Å}^3$ simulation box (box size reduced in picture). The 121,257 solvent atoms are removed for clarity.

amortize some of the data access overhead, the linked lists are packed densely and in subbox order for linear traversal and better cache locality.

For each physical dimension d, the number of subboxes, n_d, and their size, box_d, depend on several parameters including the number of processors $P(= p_1p_2p_3)$, the mapping strategy, the number of atoms N, the cutoff radius R_{cut}, and a user supplied granularity parameter. There are several tradeoffs and constraints to be observed:

- We distribute our problem domain across processors with ***subbox granularity***, *i.e.*, a certain subbox is treated as indivisible as far as ownership goes, and we assume only one owner per subbox.

  Therefore, if n_d becomes smaller, load balancing may become less accurate, since the number of different decompositions becomes smaller.

- However, if n_d becomes larger, the overhead associated with a traversal of the subboxes to locate the atoms increases.

- We also use our subbox structure to limit our search for nonbonded interaction partners of a given atom, which allows us to avoid the naïve $\mathcal{O}(N^2)$ pairlist generation algorithm. For that purpose it is advantageous if box_d is an integral fraction of R_{cut} [16].

- The hierarchical decomposition should also be able to balance the workload for the trivial case of a system with constant density. Therefore it must be possible to create subdomains of equal size; for all d, n_d should be a multiple of p_d.

Subdomains, each consisting of a connected set of subboxes, are assigned to processors according to some space-to-processor mapping strategy. Each subdomain s is associated with a

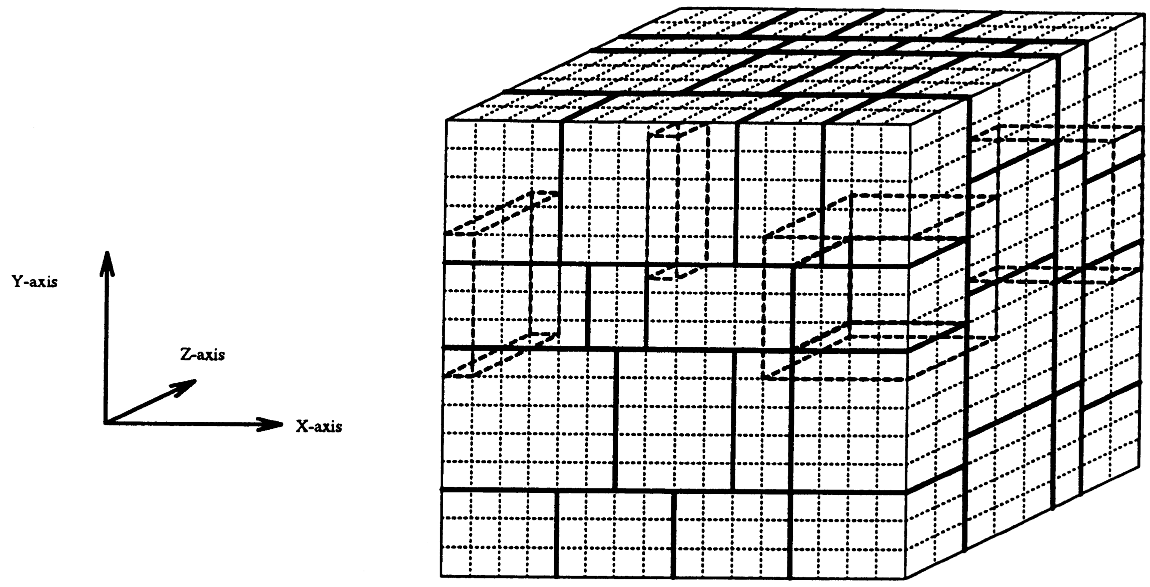


Figure 3: Subbox division of the problem domain; we use $P = p_1p_2p_3 = 4 * 4 * 4$ processors and $n_{tot} = n_1n_2n_3 = 16 * 16 * 8$ subboxes. Dotted lines indicate subboxes, heavy lines delineate processor subdomains. Hashed-lined regions show the overlap area of the processor with logical coordinate $(4, 3, 1)$, which has a subdomain consisting of $4 * 3 * 2$ subboxes located at the center of the edge closest to the reader; note the wrap-around of the overlap due to periodic boundary conditions.

certain *overlap area*, which is the set of subboxes that are not in s but reach into the cutoff radius of some subbox in s. Figure 3 shows an example configuration of subboxes and subcubes for $P = 64$ processors.

2.2 Load balancing

To allow for both inhomogeneous systems and systems that change shape over the course of a simulation, dynamic load balancing is applied to the distribution of data and computation. At the beginning of a simulation, the problem domain is first divided into p_3 slices along the z-axis, then each slice is divided along the y-axis into p_2 columns, and finally each column is divided into p_1 subdomains. This initial distribution is equivalent to a blockwise decomposition into $p_1p_2p_3$ subdomains of equal size. Every f_{bal} timesteps, each processor locally computes its own workload. (The current heuristic measures workload as the sum of the local number of atoms and the size of the local pairlist.) Given each processor's workload, a global balancing step adjusts the subdomain boundaries, assuming a homogeneous workload density within each subdomain. This is done hierarchically: first, subdomain boundaries are shifted between slices, followed by column shifts within slices and finally shifts within columns. Figure 3 shows a possible hierarchical decomposition. A smoothing factor σ is applied to avoid boundary oscillations [8].

2.3 Communication

EulerGromos communicates off-processor data accesses in one of two possible ways: *Point-to-Point* or *Shift*. In Point-to-Point communication, messages are sent directly between processor pairs that share data; in Shift communication, each processor communicates exclusively with its six immediate neighbors, relying on those to forward data to other processors it has to exchange data with. This results in a coordinated use of the interconnection network.

The Shift algorithm shifts data in three *phases*: first along the x axis, then along the y axis, and finally along the z axis. For dimension d, phase d consists of k_d subphases, $k_d = \lceil RcutU_d/procU_d \rceil$, where *procU* and *rcutU* are the subdomain extent and the cutoff-radius, respectively, in subbox units. A subphase consists of an exchange with the two neighbor processors along the axis of the current phase. Within each phase, the first subphase will shift all data received in all previous phases; within phase one, this will be the set of local atoms, the base case. For all subphases after the first subphase, a processor will communicate to each of its two neighbors the data that were received in the immediately preceding subphase from the other neighbor. Let $k_1 = k_2 = k_3 = k$; *i.e.*, let the cutoff radius penetration be uniform along the three Cartesian axes. Assuming that N atoms are uniformly distributed across P processors, then the amount of data received by each processor to exchange overlap atoms is $(8k^3 + 12k^2 + 6k)N/P$. For $k = 1$, the preceding expression evaluates to $26N/P$, the case where each processor subdomain is greater than or equal to R_{cut}.

The scalability of a communication algorithm generally depends on communication volume, bandwidth, latency, and buffering costs. However, for the problems and processor configurations considered here, the communication volume and the number of messages drive the performance (see Section 3.4). Again assuming a uniform k_d in each dimension, each processor sends $6k$ messages using Shift, compared to approximately $(2k + 1)^3$ with Point-to-Point.

Shift communication can be expected to be most efficient at large P, where the data volume communicated per processor decreases (Figure 8). However, the current implementation of the Shift communication requires subdomains of equal size, therefore it cannot be used in conjunction with the load balanced, hierarchical decomposition. Point-to-Point communication can reduce the overall communication volume to a processor, since subbox granularity is used to buffer overlap data.

2.4 Molecular dynamics

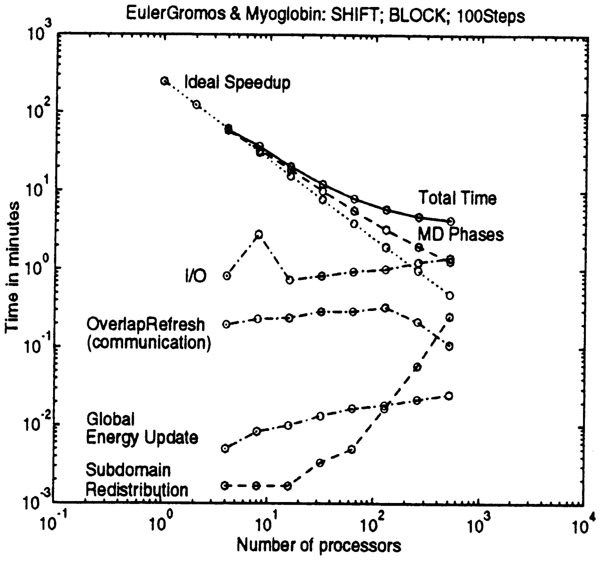
A library of routines that perform and support the Eulerian decomposition and load-balancing interface with GROMOS. The modifications to GROMOS itself were minimal, permitting reuse of at least 90% of the approximately 10,000 lines of original code. From the user's perspective, the GROMOS touch and feel are retained; I/O formats are close to identical.

With EULERGROMOS, a processor calculates all interactions involving its local atoms. The nonbonded interactions are determined geometrically, while the bonded interactions are obtained from the molecular topology. The scalability depends on the cost to determine the interactions a processor should calculate and the calculation of those interactions. To determine the nonbonded interactions, the subbox data structure is traversed to create a standard GROMOS pairlist containing the interactions. As a result, the pairlist routine required extensive modification and was rewritten. The nonbonded force routine, which uses the pairlist, was slightly modified.

Based on Newton's Third Law, GROMOS and other MD programs calculate each NBF interaction only once per pair, instead of twice [22, 9, 5]. However, for those interactions where atoms are local to different processors, the additional complexity for managing those boundary atoms could adversely effect node-level optimizations. (See [17] for further discussion.) Therefore, we chose for EULERGROMOS to use Newton's Third Law within subdomains, but not across subdomains.

Each processor extracts its local set of bonded interactions by scanning the $\mathcal{O}(N)$ global topology information. While this could limit scalability, such asymptotic limitations are outside of our practical range for N and P (see Section 3.2).

Scalability of the calculation of the potential energy (i.e., the interactions) requires a reasonable load balancing, as provided by our hierarchical load-balancing scheme (Section 3.3).



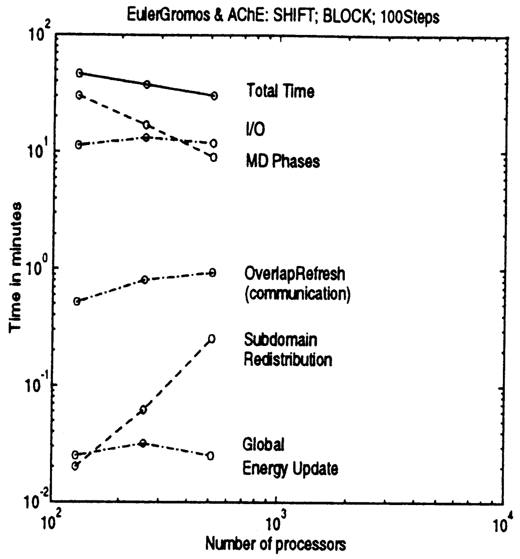


Figure 4: Performance of EULERGROMOS for two systems. Left: a 10,914 atom myoglobin system. Right: the solvated acetylcholinesterase dimer (AChE), $N = 131,663$. Note the log-log scale and the different ranges of processors. MD phases include: integration, pairlist construction, shake, bonded and nonbonded force calculations. Subdomain redistribution uses Point-to-Point communication; overlap refresh uses Shift communication. The total time includes I/O. A line with $slope = -1$ has been included for reference to ideal performance.

3 Evaluation and Experiments

3.1 The applications

We base our evaluation on two molecular simulations that are of current interest in the molecular dynamics community: solvated myoglobin, containing about 10,000 atoms, and the solvated acetylcholinesterase dimer (AChE), containing about 131,000 atoms. Performance data for $P > 8$ were acquired on the 512-node Touchstone Delta with i860 processors at Caltech; for $P \leq 8$ an iPSC/860 was used. Both machines were configured with 16 megabytes per node, however, the memory available to the application varies: the Delta provides about 12 Megabytes per node; the iPSC/860 provides about 15 Megabytes.

3.2 Overall performance

Figure 4 shows the raw performance of EULERGROMOS on myoglobin and AChE, with I/O required for production simulations included in the total time. (I/O includes coordinates and velocities output every 10 steps along with output of the final configuration.) Due to memory limitations, AChE required a minimum of 128 processors and myoglobin required four. These plots demonstrate a continuous performance improvement up to $P = 512$ for both systems, the largest machine configurations available for the experiments. The curve labeled *MD Phases* for myoglobin in Figure 4 is broken into components in Figure 5. The nonbonded and bonded force calculations approach linearity, but deviate slightly due to redundant interaction calculations and load imbalances (note that those data are from a simulation without load-balancing). However, that deviation is greater for the myoglobin system where the smaller subdomains lead to greater load imbalances and more redundant interaction calculations; the MD phases for AChE demonstrate close to ideal speedup.

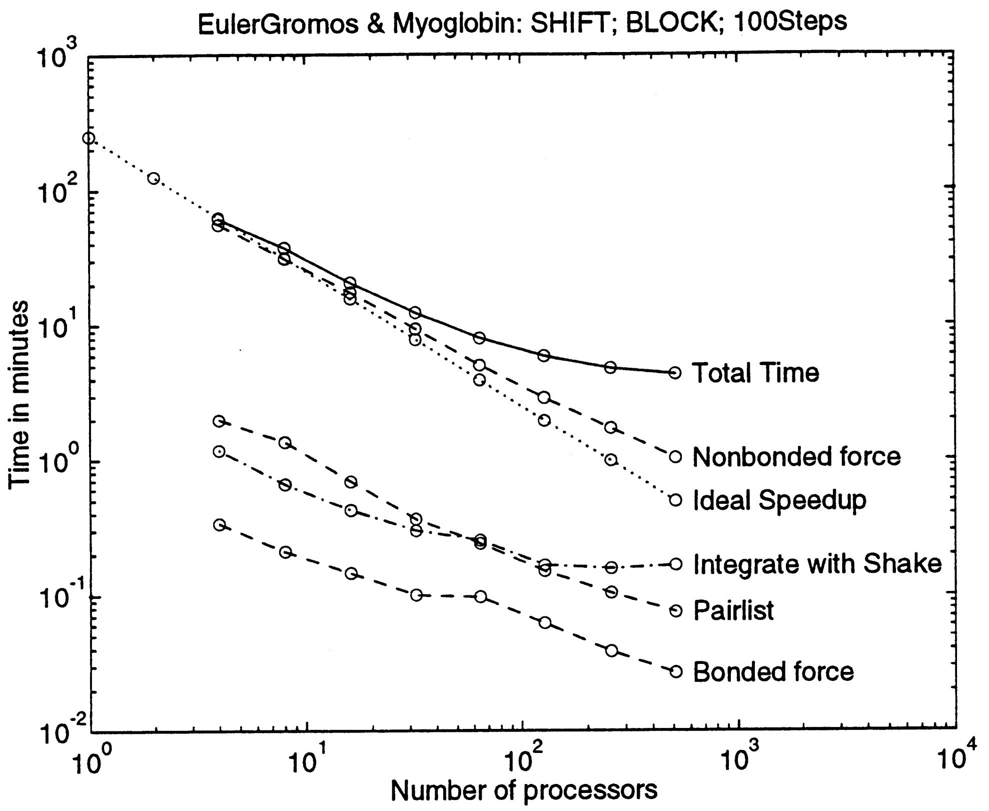


Figure 5: The wall clock times for the molecular dynamics phases within EULERGROMOS for a 10,914 atom myoglobin system. A line with *slope* $= -1$ has been included for reference to ideal performance.

3.3 Load balancing

Subboxes are viewed as indivisible with respect to subdomain boundaries. In other words, each subbox is owned by just one processor, and boundaries can be shifted only in increments of the subbox sizes. This simplifies and accelerates the mapping between atom coordinates and subdomains, but it also limits the accuracy of our load balancing: the larger the subboxes, the coarser the border shifts will be. To study this effect, we simulated myoglobin ($N = 10,914$) on 64 processors for 100 time steps with varying n_d.

The accumulated absolute border movements along each dimension increase monotonically with n_d. Borders start moving for $n_d \geq 16$. As to be expected, most activity takes place within columns, followed by movements within slices and finally across slices. Figure 6 shows the standard deviation of the workload across processors for the different n_d values. It turns out that the standard deviation decreases for higher n_d, so load balancing does achieve the desired effect.

While load balancing can be profitable for typical biomolecular systems, such as myoglobin, the advantages tend to be less pronounced there due to the density homogeneity and characteristically long time scales for large-scale motion. Consequently, the real strengths of load balancing are to be expected for systems that are inhomogeneous and change their shape within short time scales, and for varying problem domain sizes (for example, constant-pressure systems).

To study the effectiveness of load balancing for a highly inhomogeneous system, we simulated argon enlarged problem domain. As shown in Figure 7, load balancing does adapt very quickly to

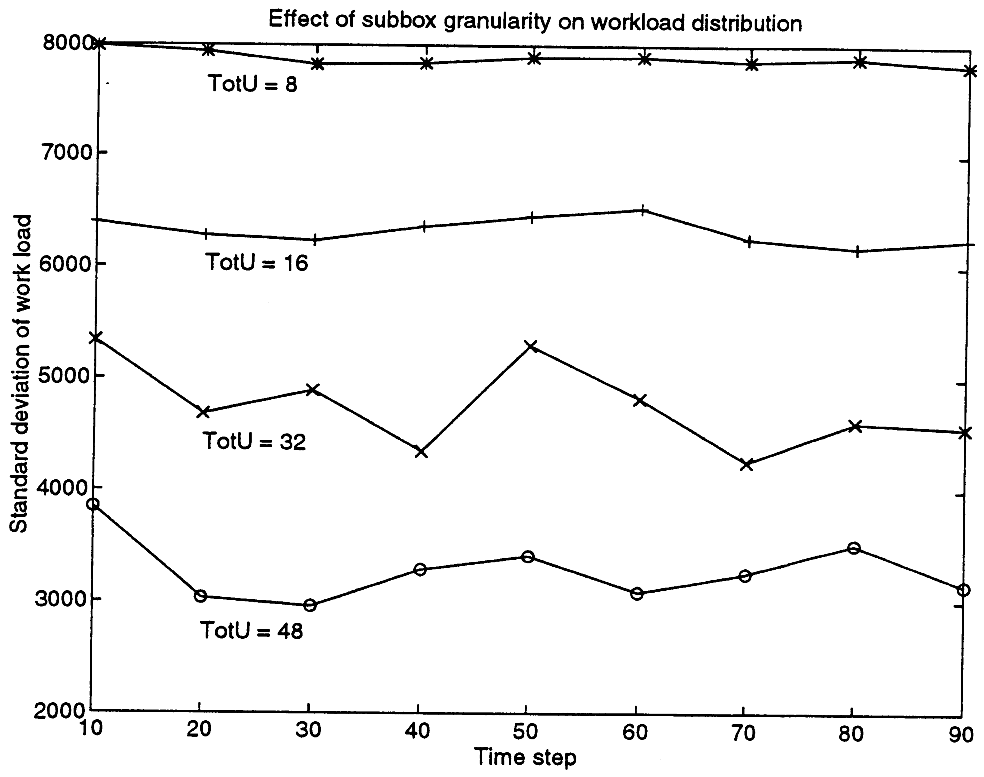


Figure 6: Effect of n_d (= TotU) on load balancedness. The mean of the work load is about $5.4 * 10^4$.

the inhomogeneity, and after three rebalancing steps we have an even workload on each processor.

3.4 Communication

Figure 8 compares Point-to-Point and Shift communication. A characteristic of the Shift algorithm is that it becomes faster as P increases, since the data communicated per processor decreases (even though the overall communication volume increases; see Section 2.3). For small P, the Point-to-Point algorithm is competitive with the Shift algorithm. However, with large P, we see that the Point-to-Point algorithm performance deteriorates exponentially.

4 Related work

There have been numerous papers on the parallelization of the link-cell method and spatial decompositions. A common feature of spatial-mapping approaches is to restrict communication to nearest neighbors; consequently, each processor subdomain has to be greater than or equal to the cutoff radius size [10, 16, 22, 9], which in turn limits scalability [21].

Esselink, et al., report a geometric decomposition where the subdomain size restriction has been lifted [5]. However, they assume a homogeneous distribution of particles with equally sized processor subdomains. Morales and Nuevo decrease the processor subdomain size such that it is less than R_{cut}, but they continue to restrict interactions to neighboring subdomains only, thereby effectively reducing R_{cut}; they evaluate the effect on thermodynamic properties [15]. (See [13] for a detailed study on cutoff radius effects.) Plimpton also allows for subdomains smaller than

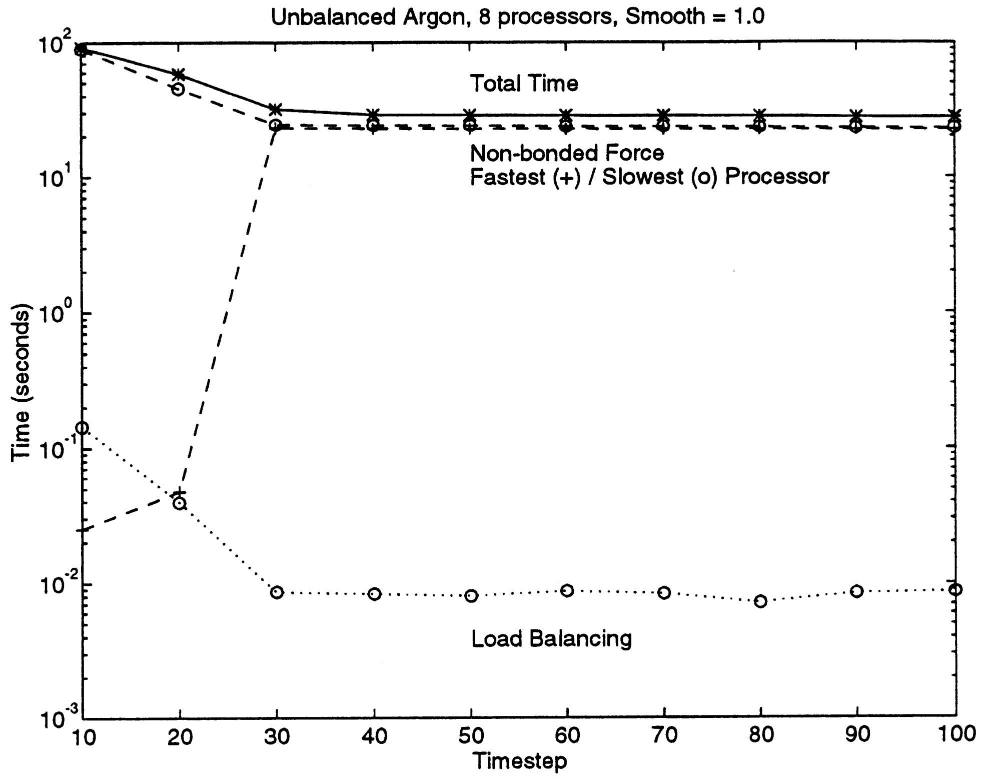


Figure 7: Effect of load balancing for Argon in an oversized box. The $N = 10,169$ argon molecules inhabit one 8Å^3 corner of a 16Å^3 periodic box. Load balancing reduces the total time per 10-step interval from 92 seconds to less than 30 seconds.

R_{cut} in an implementation reported for Lennard-Jones particles that is similar to our blockwise decomposition with the Shift algorithm [17].

5 Summary and Conclusions

This abstract gave a brief description of EulerGromos, a parallel molecular dynamics program, and evaluated its performance characteristics. The measurements reported here indicate that the main design goal, scalability, has been achieved: Increasing the number of processors results in additional speedups both for fixed and growing problem sizes. In particular, we have achieved performance increases with as few as 20 atoms per processor, much less than could be achieved with subdomains at least as large as the cutoff radius.

Scalably extending the calculation over processors reduces the cost per timestep so that within some fixed simulation period

- phase space sampling is more extensive, and
- larger systems with better boundary conditions can be simulated.

EulerGromos has also simulated systems with over 100,000 atoms, about an order of magnitude larger than usual solvated biomolecule simulations.

To summarize, the major features of EulerGromos are:

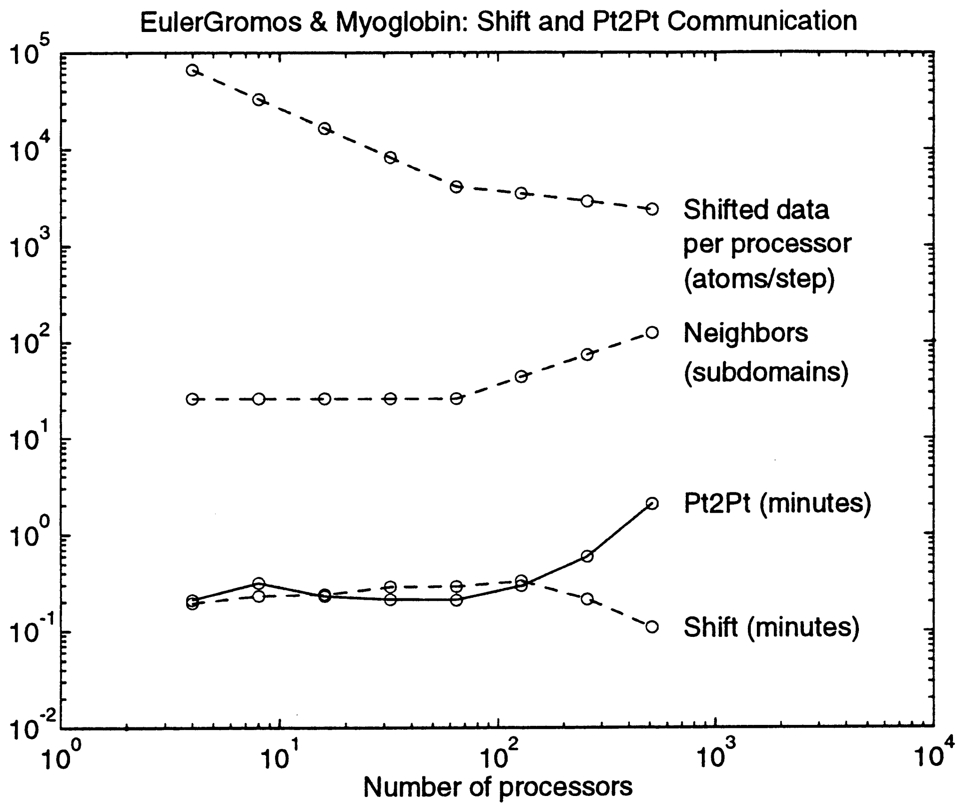


Figure 8: Shown are: the volume communicated per processor per time step for the shift algorithm in units of atoms (calculated based on uniform distribution), the number of processor subdomain neighbors for both algorithms, and the times spent in the overlap refresh phase for 100 steps of molecular dynamics with myoglobin.

- A flexible data structure with load balancing.
- Efficient communication algorithms.
- Low overheads, also relative to small workloads.

Acknowledgements

We thank Stan Wlodek and Yuan Cai for programming help.

Program development was performed on an iPSC/860 provided by Intel and housed at the University of Houston, Institute for Molecular Design. Delta simulations were performed on the Intel Touchstone Delta System operated by Caltech on behalf of the Concurrent Supercomputing Consortium with access provided by the Center for Research on Parallel Computation.

This work was supported in part by the National Science Foundation, award number ASC-921734 (which includes funds from DARPA) and the R.A. Welch Foundation.

The work of v. Hanxleden was supported by an IBM fellowship, and by the National Aeronautics and Space Administration/the National Science Foundation under grant #ASC-9349459.

References

[1] Bernard R. Brooks, Robert E. Bruccoleri, Barry D. Olafson, David J. States, S. Swaminathan, and Martin Karplus. CHARMM: A program for macromolecular energy, minimization and dynamics calculations. *Journal of Computational Chemistry*, 4(2):187–217, 1983.

[2] Bernard R. Brooks and Milan Hodoscek. Parallelization of CHARMM for MIMD machines. *Chemical Design Automation News*, 7(12):16–22, 1992.

[3] T. W. Clark and J. A. McCammon. Parallelization of a molecular dynamics non-bonded force algorithm for MIMD architectures. *Computers & Chemistry*, 14(3):219–224, 1990.

[4] J. Eastwood, R. Hockney, and D. Lawrence. PM3DP – the three dimensional periodic particle-particle/particle-mesh program. *Computer Physics Communications*, 19:215–261, 1977.

[5] K. Esselink, B. Smit, and P.A.J. Hilbers. Efficient parallel implementation of molecular dynamics on a toroidal network. Part II. Multi-particle potentials. *Journal of Computational Physics*, 106:101–107, 1993.

[6] David Fincham. Parallel computers and molecular simulation. *Molecular Simulation*, 1:1–45, 1987.

[7] W. F. van Gunsteren and H. J. C. Berendsen. GROMOS: GROningen MOlecular Simulation software. Technical report, Laboratory of Physical Chemistry, University of Groningen, Nijenborgh, The Netherlands, 1988.

[8] R. v. Hanxleden and L. R. Scott. Load balancing on message passing architectures. *Journal of Parallel and Distributed Computing*, 13:312–324, 1991.

[9] Fredrik Hedman and Aatto Laaksonen. Data parallel large-scale molecular dynamics for liquids. *International Journal of Quantum Chemistry*, 46:27–38, 1993.

[10] Tsutomu Hoshino and Kiyo Takenouchi. Processing of the molecular dynamics model by the parallel computer PAX. *Computer Physics Communications*, 31(4), 1984.

[11] M. Karplus and J. A. McCammon. Dynamics of proteins: Elements and function. *Ann. Rev. Biochem.*, 53:263–300, 1983.

[12] S. L. Lin, J. Mellor-Crummey, B. M. Pettitt, and G. N. Phillips, Jr. Molecular dynamics on a distributed-memory multiprocessor. *Journal of Computational Chemistry*, 13(8):1022–1035, 1992.

[13] Richard J. Loncharich and Bernard R. Brooks. The effects of truncating long-range forces on protein dynamics. *PROTEINS: Structure, Function and Genetics*, 6:32–45, 1989.

[14] J. A. McCammon and Stephen C. Harvey. *Dynamics of proteins and nucleic acids.* Cambridge University Press, Cambridge, 1987.

[15] Juan J. Morales and Maris J. Nuevo. A technique for improving the link-cell method. *Computer Physics Communications*, 60:195–199, 1990.

[16] M. R. S. Pinches and D. J. Tildesley. Large scale molecular dynamics on parallel computers using the link-cell algorithm. *Molecular Simulation*, 6:51–87, 1991.

[17] Steve Plimpton. Fast parallel algorithms for short-range molecular dynamics. Technical Report SAND91-1144, Sandia National Laboratories, Albuquerque, New Mexico 87185, May 1993.

[18] A. Rahman. Correlations in the motion of atoms in liquid argon. *Physical Review*, 136(2A):405–411, 1964.

[19] D. C. Rapaport. Multi-million particle molecular dynamics, I. Design considerations for vector processing. *Computer Physics Communications*, 62:198–210, 1991.

[20] Jean-Paul Rycaert, Giovanni Ciccotti, and Herman J.C. Berendsen. Numerical integration of the cartesian equations of motion of a system with constraints: Molecular dynamics of n-alkanes. *Journal of Computational Physics*, 23:327–341, 1977.

[21] Hiroyuki Sato, Yasumasa Tanaka, Hiroshi Iwama, Shigetsugu Kawakida, Minoru Saito, Kenji Morikami, Toru Yao, and Shigenobu Tsutsumi. Parallelization of AMBER molecular dynamics program for the AP1000 highly parallel computer. In *Scalable High Performance Computing Conference*, Williamsburg, VA, 1992.

[22] W. Smith. Molecular dynamics on hypercube parallel computers. *Computer Physics Communications*, 62:229–248, 1991.