John von Neumann Institute for Computing

A Load Balanced Force-Domain Decomposition Algorithm for Parallel Molecular Dynamics Simulations

G. Sutmann, F. Janoschek

published in

From Computational Biophysics to Systems Biology (CBSB07),
Proceedings of the NIC Workshop 2007,
Ulrich H. E. Hansmann, Jan Meinke, Sandipan Mohanty,
Olav Zimmermann (Editors),
John von Neumann Institute for Computing, Jülich,
NIC Series, Vol. **36**, ISBN 978-3-9810843-2-0, pp. 279-282, 2007.

http://www.fz-juelich.de/nic-series/volume36

A Load Balanced Force-Domain Decomposition Algorithm for Parallel Molecular Dynamics Simulations

Godehard Sutmann[1] **and Florian Janoschek**[2]

[1] Central Institute for Applied Mathematics and
John von Neumann Institute for Computing,
Research Centre Jülich, 52425 Jülich, Germany
E-mail: g.sutmann@fz-juelich.de

[2] Stuttgart University
Institute for Computational Physics, Pfaffenwaldring 27
D - 70569 Stuttgart, Germany

1 Introduction

Classical molecular dynamics simulations are often considered as the *method par excellence* to be ported to parallel computers, promising a good scaling behavior. On the one hand parallel algorithms exist which enable good scaling[1]. On the other hand the complexity of the problem at hand, often scales like $\mathcal{O}(N)$, enabling a linear increase of problem size with memory. However, this point of view applies only to a limited class of problems which can be tackled by molecular dynamics. E.g. in the case of homogeneous periodic systems, where particles interact via short range interactions, the most efficient algorithm is a domain decomposition scheme, guaranteeing local communication between processors and therefore allowing good parallel scaling. In combination with linked-cell lists, the problem scales like $\mathcal{O}(N)$ both in computational complexity and memory demand, so that an ideal behavior in both strong and weak scaling might be expected.

On the other hand, this ideal behavior breaks down if different problem classes are considered, e.g. the case of long range interactions, where not only local communications between processors are required. Another class of counter examples is the case of inhomogenous systems, which occur e.g. in open systems, where the particle density is considerably larger in the center of the system than in the diffuse halo or e.g. in systems consisting of different thermodynamic phases as is the case for the coexistence of liquid/gas or solid/gas phases. In this case, domain decomposition algorithms often fail. Due to a more or less regular geometric decompoition of space, processors are responsible for different numbers of particles, often introducing a strong load-imbalance, which leads to inefficient CPU usage on some processors and therefore to a bad parallel scaling.

For these classes, other parallel decomposition schemes are often applied, like atom- or force-decomposition schemes[1]. While the former one distributes equally particles onto processors, the latter one decomposes the interaction matrix among processors. It is the latter case which is considered in the present paper. This method also allows a geometric approach which consists of equally partitioning of the force-matrix $\mathsf{F} \in \mathbb{R}^{3N\times 3N}$ onto processors. Taking into account Newton's law of action and counteraction either the upper triangular part of F may be decomposed into equal areas[2], or the whole matrix is decomposed[3,4], while assigning only a subset of interactions onto each PE, in order to fulfil the skew symmetry of the interactions.

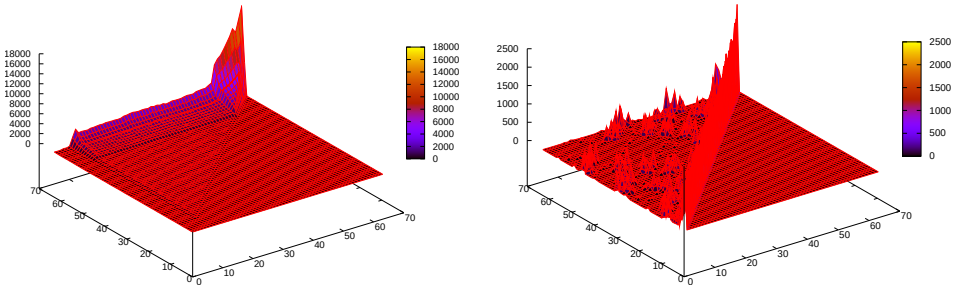

Figure 1. Communication patterns between processors for random distribution of particles (left) and sorted particles according to a space-filling Hilbert curve (right). Note the different magnitudes of data volume. Shown is the case for $P = 64$. Left part of each figure shows the upper triangular part of the force matrix. Each row corresponds to data stored on individual PEs.

In general there are two points in force decomposition methods, where communication is required:

1. after propagating the particle position in the integration step, the position of particles must be transferred to remote processors in order to calculate mutual interactions.

2. after calculating interactions, the partial forces, calculated on different processors acting on a tagged particle must be collected onto the host processor of that particle, in order to propagate its positions and momenta.

2 Method

2.1 Communication

Traditionally, coordinates and forces are transferred by all-to-all communication steps. Positions are usually distributed by an `mpi_allgatherv` command while forces are summed up simply by an `mpi_allreduce` procedure. This is certainly the most simple way to proceed. Since most MPI implementations internally make use of a tree-wise communication protocol, the global communication will scale like $\mathcal{O}(\log_2(P))$, if P is the number of processors. However, for a big system and a large number of processors, there will be a lot of redundant data transferred to processors. I.e. global communication operations do not take into account whether transferred data are really needed on remote processors. Therefore two alternative methods are considered here.

The first approach still gathers all position coordinates from remote PEs, in order to calculate interactions with local particles. However, the forces are selected according to whether they have been calculated or not. This avoids sending a lot of redundant information across the network, although the communication protocoll gets a little bit more involved. In the case of the upper triangular force matrix decomposition, forces must be sent from local PEs to the ones with larger rank. In this case $P - i_p - 1$ communication steps have to be performed, if i_p is the rank of the local PE. In this case two `mpi_gatherv` operations have to be performed: (i) to transfer the non-zero partial forces, (ii) to transfer the particle indices, onto which forces act.

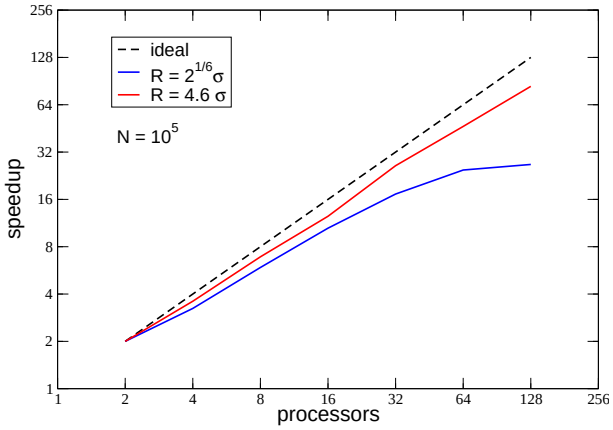

Figure 2. Parallel speedup for the force-domain decomposition method for the case of $N = 10^5$ particles and two different cutoff radii for interparticle interactions.

The second approach is more involved. The basic principle is to combine a domain decomposition with a force decomposition scheme. Domain decomposition is achieved by sorting the particles according to their positions along a space filling Hilbert curve. This ensures that most particles which are local in space are also local in memory. For short range interactions this implies that most interaction partners are stored already on the same PE. According to this organisation, the force matrix becomes dominant around the diagonal and off-diagonal areas are sparse. The next step is to calculate interaction lists, which are distributed onto all PEs. Indices of interaction partners are stored in Verlet lists, which are valid for a number of time steps. Therefore, these lists only have to be created and distributed from time to time (e.g. every 20 time steps[5]). According to these lists it is known which particles from remote PEs are needed on local PEs. The same is true for calculated partial forces. Since the amount of data is very small with respect to global communications of positions and forces, the communication overhead of this method is strongly reduced, enabling a very much better parallel scaling. Since randomisation of particle positions occur on a diffusive time scale, the space filling curve has to be updated only one or two orders of magnitude less than the interaction lists, thus introducing only a small overhead.

2.2 Load-Balancing

Another contribution of the present paper is to combine the proposed methods with an efficient load-balancing strategy. This is developed for the force-stripped row method, which partitiones equal areas in the interaction matrix to different processors. For inhomogenous systems, this approach becomes quite inefficient, since the number of interactions within such areas may vary considerably. Therefore, two different approaches are discussed and implemented here. The first consists in distributing the number of interactions equally onto the pocessors. This may give good results for the case, where particles are located randomly in memory. However, if a subset is random and another subset is ordered (as it might occur for the case of a solid/liquid system) in memory, cache effects will destroy the

equal load on PEs. Therefore, a second approach consists in distributing the work, proportional to the time which is spent on every PE. This strategy turns out to be most efficient, leading to a very fast distribution of equal load between the processors.

3 Results

The above methods are tested for a simple system consisting of Lennard-Jones particles. Molecular dynamics simulations were carried out for systems with $N = 10^5$ particles. Fig. 1 shows the amount of data which has to be communicated between processors for the cases of traditional force-decomposition (FD) methods and the force-domain-decomposition (FDD) technique, which uses the space filling Hilbert curve. In FD interacting particles are randomly distributed in memory (at least for long simulations particles get uncorrelated). Therefore, coordinates have to be sent from a processor p_i to all other processors $p_j < p_i$, while forces have to be redistributet in the opposite direction from p_j to all other processors $p_i > p_j$. Sorting reduces significantly the amount of data to be sent. First of all the size of the data buffers become smaller, second the matrix becomes sparse, i.e. a given processor p_i does not send coordinates to all other processors $p_j < p_i$, but only to those where these coordinates are really needed to compute interactions. Since sorting concentrates interaction partners close to the diagonal, only small amounts of data have to be sent to remote processors. Calculations were carried out for the FDD method, considering a system of $N = 10^5$ particles with different size of the interaction radius R_c. For decreasing R_c, the amount of data to be sent to remote PEs gets smaller. On the other hand also the work performed on a single PE decreases, because of a smaller number of interaction evaluations. Therefore, for small R_c communication becomes a bottleneck for a large number of processors, because of the accumulation of latency. Therefore, the speedup curve saturates for 128 processors. In the case of a larger cutoff ($R_c = 4.6\,\sigma$) the computation dominates communication and the program scales up to 128 processors. Note, that for a global communication, like in traditional FD, communication would have a significant larger contribution and the program would not scale as well as in the present case.

References

1. S. Plimpton. Fast parallel algorithms for short range molecular dynamics *J. Comp. Phys.*, 117:1, 1995.
2. R. Murty and D. Okunbor. Efficient parallel algorithms for molecular dynamics simulations. *Parall. Comp.*, 25:217–230, 1999.
3. V. E. Taylor, R. L. Stevens, and K. E. Arnold. Parallel molecular dynamics: Communication requirements for parallel machines. In *Proc. of the fifth Symposium on the Frontiers of Massively Parallel Computation*, pages 156–163, 1994.
4. S. Plimpton and B. Hendrickson. A new parallel method for molecular dynamics simulation of macromolecules. *J. Comp. Chem.*, 17:326, 1996.
5. G. Sutmann and V. Stegailov. Optimization of neighbor list techniques for molecular dynamics simulations. *J. Mol. Liq.*, 125:197–203, 2006.