Towards an efficient implementation of PME on low-bandwidth Linux clusters

C. Kutzner, B. de Groot, H. Grubmüller, Max-Planck-Institute for Biophysical Chemistry, Theoretical and Computational Biophysics Department, Göttingen

in collaboration with D. van der Spoel (Uppsala University), E. Lindahl (Stockholm University) and J. Pichlmeier (IBM Munich)

Abstract / poster guide

Particle-Mesh-Ewald[2,3,4] (PME) is an efficient method for calculating the long-range part of the electrostatic forces in molecular dynamics (MD) simulations. A protein embedded in a box of water molecules is a typical example of such an MD system which then consists of several 100 000s of atoms. A numerical simulation that covers a significant time-span (to reveal protein unfolding for example) can take weeks to months even on a modern parallel computer. Parallel PME involves a lot of inter-process communication and therefore soon turns out to be the bottleneck, particularly for larger CPU numbers. The GROMACS[1] MD simulation program scales reasonably well up to 20 CPUs on computers with a fast interconnect like the IBM Regatta. On a standard Linux cluster with **gigabit ethernet** however, scaling breaks down already at 4 or 8 CPUs (**Fig. 9, black line**).

Our goal is to enhance the PME performance on parallel computers without expensive network hardware. Therefore two modifications of the algorithm are being implemented that both minimize the time expensive inter-process communication:

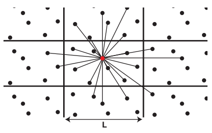
A. The atom decomposition which is used in GROMACS (**Fig. 1**) is changed into a domain decomposition for the PME part (**Fig. 2**). This way less communication is needed and time gained that clearly outweighs the time spent for reordering the atoms (**Fig. 9, green line**).

B. PME is moved to a processor subgroup while the remaining CPUs calculate the short-range interactions (**Fig. 3**). The parallel FFT, which is required twice for each PME step, is known to scale much better on a smaller number of processors. At a high number of CPUs the time gain during the FFTs outweighs the time loss for transferring the particle data between the processor groups (**Fig. 9, blue line**). Small MD systems gain speedup with PME/PP division even on a small number of CPUs (**Figs. 7-8**).

Particle-Mesh-Ewald (PME)

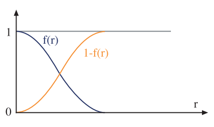
Consider a periodic system (box length L) with N particles (charges q_i) at positions r_i. To evaluate the Coulomb forces the electrostatic potential V is needed:

$$V = \frac{1}{2}\sum_{i,j=1}^{N}\sum_{\mathbf{n}\in\mathbb{Z}^3}{}' \frac{q_i q_j}{|\mathbf{r}_{ij}+\mathbf{n}L|}$$



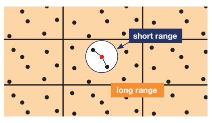
A straightforward summation is impracticable, however, the problem can be separated into two parts with the use of

$$\frac{1}{r} = \underbrace{\frac{f(r)}{r}}_{\text{short range}} + \underbrace{\frac{1-f(r)}{r}}_{\text{long range}}$$



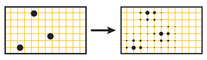
V then consists of a short range part V_{dir} and a long range part V_{rec}, both exponentially converging:

$$V = \underbrace{V_{dir}}_{\text{direct space}} + \underbrace{V_{rec}}_{\text{fourier space}}$$



V_{dir} is directly summed up to some cutoff radius, V_{rec} is evaluated in Fourier space. Since V_{rec} varies slowly, only the first couple of Fourier vectors are needed.

V_{rec} needs the Fourier transformed charge density. To be able to use the discrete Fourier transformation (FT) the charges are spread on a grid:



Each charge is distributed among its 4x4x4 = 64 neighbouring grid points. The charge mesh has a spacing of 0.12 nm and is a good approximation for the distribution of charges at the particle positions.

The Coulomb forces are evaluated in the following way each time step (The color scheme corresponds to that of Figs. 4-8):

a Calculate short-range Coulomb forces $\mathbf{F}_{i,dir}$

b Calculate long-range Coulomb forces with PME:

1 put charges on grid

2 FFT charge grid convolution FFT back
yields potential V_{rec} at grid points

3 interpolate grid and derive forces at atom positions
$\mathbf{F}_{i,rec} = -d/d\mathbf{r}_i V_{rec}$

GROMACS 3.2.1 parallel PME

The original implementation of the parallel PME algorithm requires for each CPU one MPI_Reduce operation to sum the individual contributions to the charge grid. After the FFT one MPI_Bcast per CPU is done to broadcast the potential grid to the other CPUs. Fig. 4 shows how time-expensive these collective operations are.

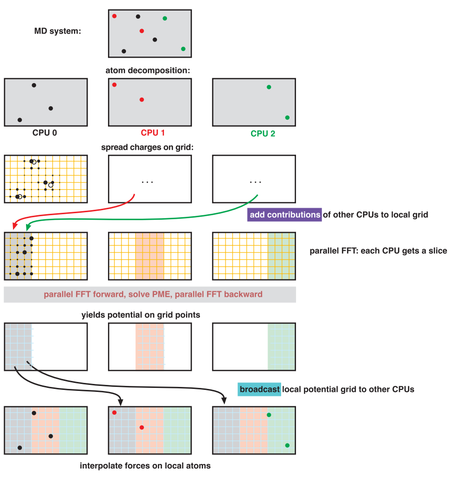
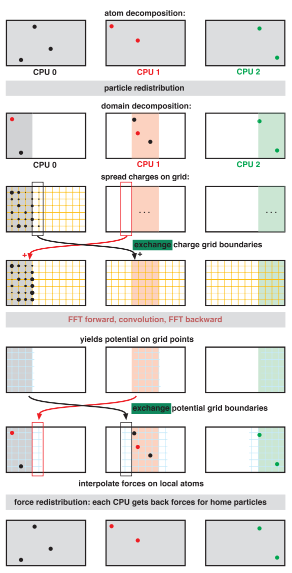

MD system:
atom decomposition:
CPU 0
CPU 1
CPU 2
spread charges on grid:
add contributions of other CPUs to local grid
parallel FFT: each CPU gets a slice
parallel FFT forward, solve PME, parallel FFT backward
yields potential on grid points
broadcast local potential grid to other CPUs
interpolate forces on local atoms

Fig. 1: Original parallel PME scheme as implemented in GROMACS 3.2.1.

A. Domain decomposition in PME part

The first improvement is to change the atom decomposition into a domain decomposition for the PME part.


atom decomposition:
CPU 0
CPU 1
CPU 2
particle redistribution
domain decomposition:
spread charges on grid:
exchange charge grid boundaries
FFT forward, convolution, FFT backward
yields potential on grid points
exchange potential grid boundaries
interpolate forces on local atoms
force redistribution: each CPU gets back forces for home particles

After the spreading of the charges, only the domain boundaries have to be communicated (2 MPI_Sendrecv operations, independent of CPU number!)

Fig. 2: Parallel PME with domain decomposition as implemented by J. Pichlmeier

B. Splitting of CPUs into PME and PP

The second improvement is to additionally split the parallel computer into a group of CPUs which calculate the short range (Particle-Particle, PP) Coulomb forces and a group of CPUs which do the long range (PME) part.

At the start of a time step, the PP CPUs send the particle data to the PME CPUs which at the end of the time step send back the long-range forces.

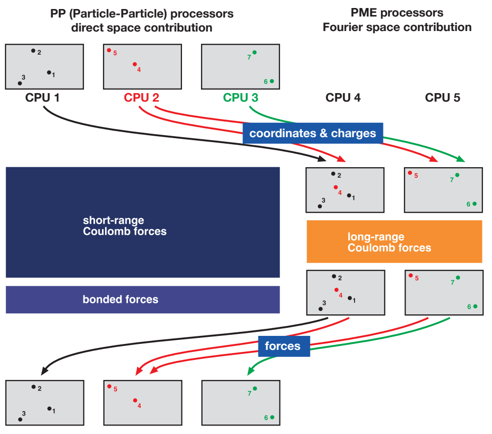

PP (Particle-Particle) processors
direct space contribution
PME processors
Fourier space contribution
CPU 1
CPU 2
CPU 3
CPU 4
CPU 5
coordinates & charges
short-range Coulomb forces
long-range Coulomb forces
bonded forces
forces

Fig. 3: PME/PP division scheme. Each PP CPU sends its particle data directly to the PME CPU which needs it to process the long-range Coulomb forces.

Gromacs 3.2.1 time step (4 CPUs)

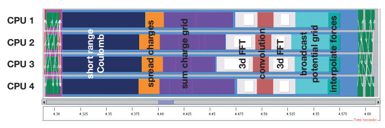


Fig. 4: Analysis of a single time step: Time (s) on horizontal axis, CPU number on vertical axis. CPUs: Intel 3.06 GHz Dual Xeons, network: gigabit ethernet, MD system consists of approx. 80 000 atoms.

Domain decomposition time step (4 CPUs)

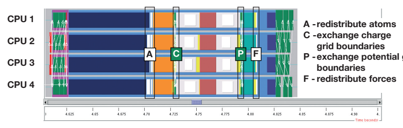


Fig. 5: Same as Fig. 4, with PME domain decomposition. The time step is much shorter, as the summation of the charge grids and the broadcast of the potential grid (compare Fig. 4) were replaced by exchange of grid boundaries (C, P).

PME/PP splitting time step (4 CPUs)

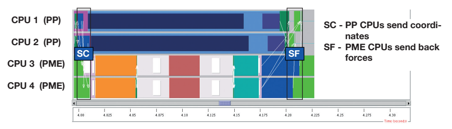


Fig. 6: Same as in Fig. 4, but with domain decomposition and PME/PP splitting. Short-range forces are done by PP CPUs 1 and 2, long-range by PME CPUs 3 and 4.

A small test system (4 000 atoms, 6 CPUs)

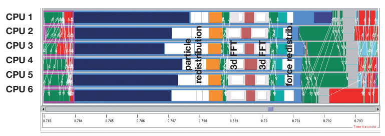


Fig. 7: Time step for a 4 000 atom system with domain decomposition only. Speedup on 6 CPUs is **2.8**. A significant part of the time is lost by redistributing the particles and forces.

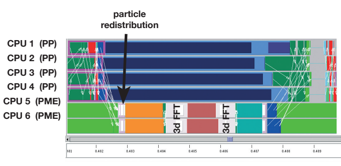


Fig. 8: Same as Fig. 7, but with domain decomposition and PME/PP splitting. A speedup of **3.6** can be achieved, because particle redistribution is much faster on just 2 CPUs while the FFT is not much slower.

Speedup comparison (80 000 atoms)

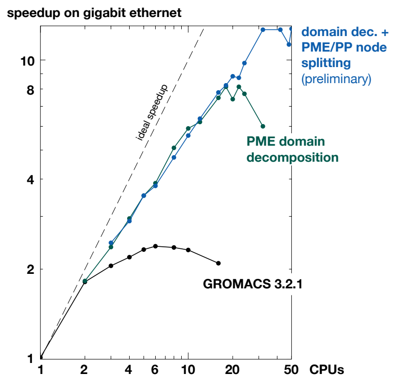


Fig. 9: Speedups for different PME implementations on Intel Dual Xeons 3.06 GHz with gigabit ethernet. The test system is a protein embedded in a lipid bilayer membrane surrounded by water and consists of approx. 80 000 atoms.

Summary

Achievements for gigabit ethernet:

- with domain decomposition:
  scaling @ 4 CPUs & 80 000 atoms **0.74 !** (was 0.54)
- domain decomposition and node splitting:
  speedup of > 10 possible ! (was: 2.5)
- **PME/PP splitting** necessary for **small systems** on **many CPUs**
- Outlook: speed-up splitting code (e.g. by using non-blocking communications)

References

(1) E. Lindahl, B. Hess, D. van der Spoel (2001): GROMACS 3.0: a package for molecular simulation and trajectory analysis, J. Mol. Model., 7.

(2) T. Darden, D. York, L. Pedersen (1993): Particle mesh Ewald: An N·log(N) method for Ewald sums in large systems, J. Chem. Phys. 98, 12.

(3) U. Essmann, L. Perera, M. L. Berkowitz, T. Darden, H. Lee, G. Pedersen (1995): A smooth particle mesh Ewald method, J. Chem. Phys. 103, 19.

(4) M. Deserno, C. Holm (1998): How to mesh up Ewald sums. I. A theoretical and numerical comparison of various particle mesh routines, J. Chem. Phys. 109, 18.