AMOEBA and HPC
Parallelising Polarisable Force Fields

Weronika Filinger (w.filinger@epcc.ed.ac.uk)
EPCC, The University of Edinburgh

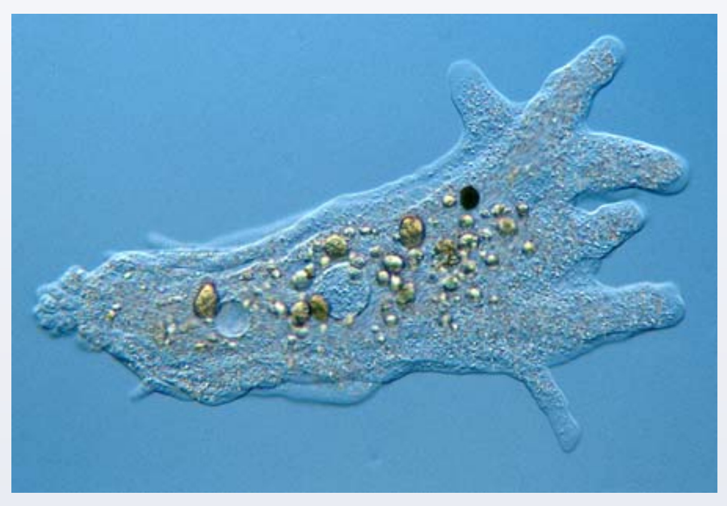
Why we want to use polarisable force fields

The accuracy of bio-molecular simulations can be seriously limited by the use of traditional treatment of electrostatics from empirically-derived force fields. Non-polarisable models typically approximate electrostatic interactions by only using fixed, usually atom-centred, charges, so that the charges in atoms are limited in their ability to adapt to changes in their local environment. Real physical systems, however, do adapt and undergo polarisation, e.g. the orientation of atom bonds will distort when placed in poorly conducting mediums such as water, changing the geometry, energy and the way the system interacts. Polarisation is thus a necessary and important intermolecular interaction required to accurately model chemical reactions in biological molecules and water-based systems, which are of great interest in fields such as chemistry, biochemistry and material science.

Much effort has been put into developing mutually polarisable models, which enhance fixed charge models by incorporating the polarisation of atoms and molecules in the dynamics of such systems. These polarisable models are more accurate and more complex hence they tend to be much more computationally demanding. Therefore there is a strong requirement for the software packages that implement polarisable force fields to be capable of exploiting current and emerging HPC architectures to make modelling ever more complex systems viable.

AMOEBA

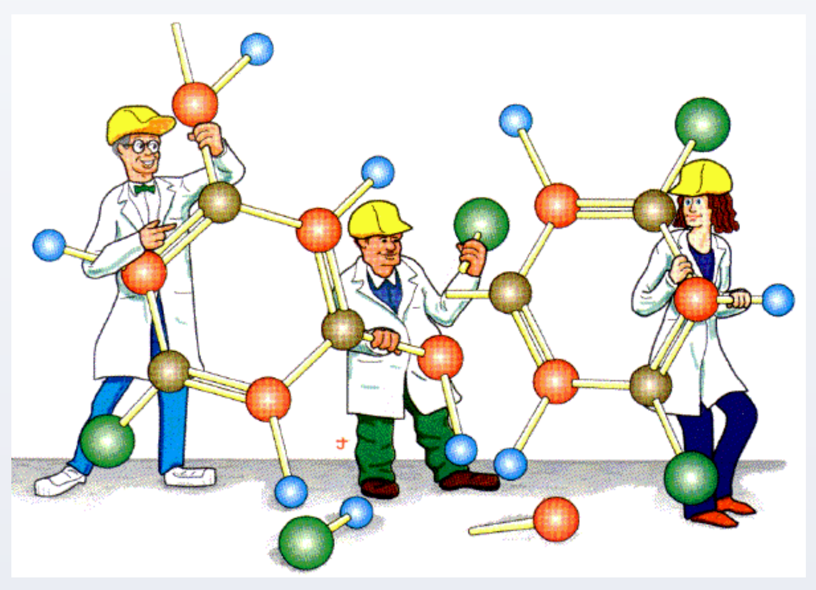
AMOEBA [1] (**A**tomic **M**ultipole **O**ptimised **E**nergies for **B**iomolecular **A**pplications) is a polarisable force field that has been implemented and is used in codes such as TINKER [2] and AMBER [3]. AMOEBA replaces fixed partial charges with atomic multipoles, including an explicit dipole polarisation that allows atoms to respond to the polarisation changes in their molecular environment.

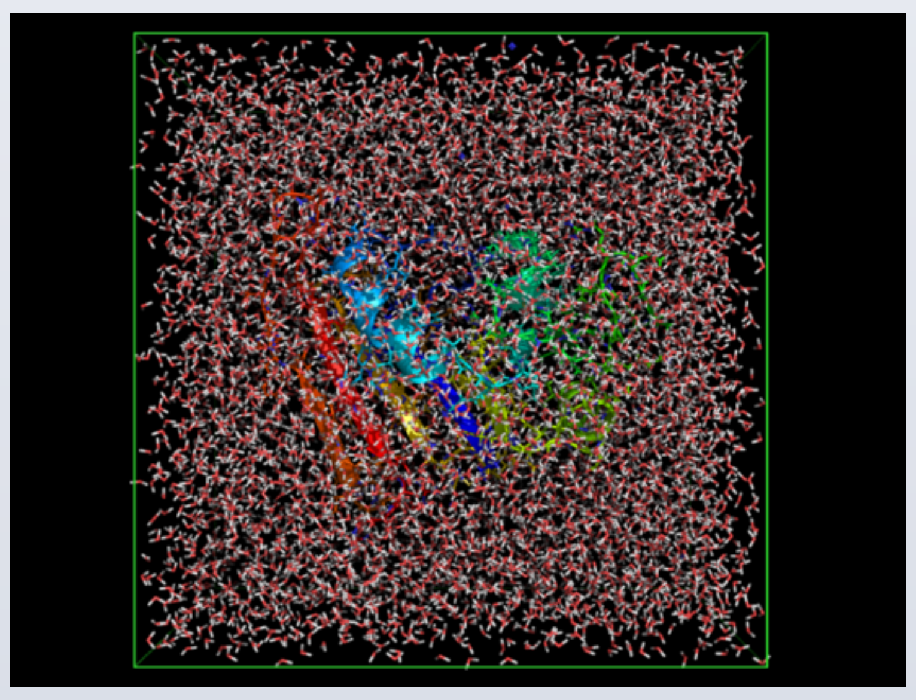

Amoeba proteus
Image taken from http://www.microscopy-uk.org.uk/mag/imgsep01/amoebaproteus450.jpg

AMOEBA has been shown to give more accurate structural and thermodynamic properties of various simulated systems [4]. However, the time required to calculate the effect of the induced dipoles alone is 15-20 times larger than the cost of one time step for a fixed charge model. If this computational time could be reduced then the AMOEBA force field could, for example, model a much broader range of protein-ligand systems than is currently possible or provide a means of refining the analysis of X-ray crystallography data to cope with large and challenging data sets, such as for ribosome crystals. The larger computational costs associated with the polarisation calculations needs to be mitigated by improving the parallelisation of codes such as TINKER.

TINKER

TINKER is a software package capable of molecular mechanics and dynamics, with some special features for biopolymers, written by Jay Ponder (Washington University in St. Louis). TINKER consists of a collection of programs offering different functionalities and supports customised force fields parameter sets for proteins, organic molecules, inorganic systems, etc., including the AMOEBA force field.


TINKER's logo taken from TINKER's website[2]


DHFR in water used in JAC benchmark
Image taken from http://www.gromacs.org/@api/deki/files/130/=dhfr_water.png

TINKER is written in Fortran and has been parallelised using OpenMP directives. Nearly 95% of the ***dynamic*** program included in the Tinker distribution, which can be used to generate a stochastic dynamic trajectory or compute a molecular dynamics trajectory in a chosen ensemble, can be executed in parallel. However, the scalability of ***dynamic*** has been found to be limited. Hence further parallelisation efforts have been undertaken by EPCC as part of the **A**dvanced **P**otential **E**nergy **S**urfaces (APES) project, a NSF-EPSRC funded US-UK collaboration.

OpenMP Performance

The performance of ***dynamic*** on the Joint AMBER-CHARMM (JAC) benchmark [5] run on the UK ARCHER supercomputing service is shown in Figure 1. Each ARCHER node has 24 cores so the program can be run on up to 24 OpenMP threads. The JAC system consists of 2489 atoms of dihydrofolate reductase (dhfr) in a bath of 7023 water molecules, giving a total of 23,558 atoms. The simulation run 100 MD steps in a canonical ensemble (NVT - constant number of atoms, volume and temperature) at a temperature of 298 Kelvins. The length of a time step was set to 1.0 femtoseconds.

From the performance and scalability analysis we can note that:

- The application scales up for to 20 OpenMP threads but a parallel efficiency above 50% is only observed for up to 12 threads;
- Slightly better performance and scalability are observed for larger systems, but memory usage restricts the maximum size of the system that can be run;
- An attempt to improve the performance through OpenMP optimisations were not very successful; small performance improvements were compiler and architecture dependant;
- The difference between the observed and theoretical speedups suggests that without increased parallel coverage the OpenMP optimisation will not improve the performance significantly;

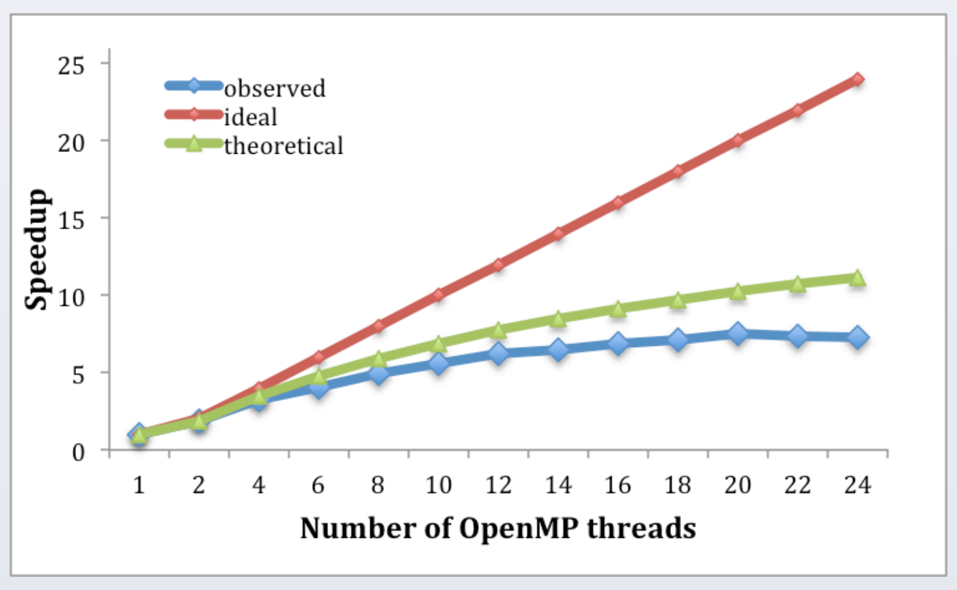


Figure 1. Speedups for different number of OpenMP threads for JAC benchmark on ARCHER.

The *ideal speedup* is the speedup that would be obtained if the whole code was parallelised with maximal efficiency, and the *theoretical speedup* is the speedup calculated using Amdahl's law [6] with the serial component (i.e. the fraction of the code that is executed serially) of 5%.

The next step is to create a hybrid MPI and OpenMP implementation to make use of the shared memory within a node via OpenMP and to use MPI for the internode communications.

MPI Parallelisation

An MPI parallelisation effort is under way. Due to the time constraints of the APES project and the complexity of the Tinker code EPCC's efforts have been focused on a replicated data strategy. This should enable performance improvements within the time frame of the project.

MPI processes	1	2	4	8	16
Speedup	1	1.48	1.85	1.87	1.36
Parallel Efficiency	100%	74%	46%	23%	9%

Table 1. Speedup and parallel efficiency for different number of MPI processes for JAC benchmark.

Micro-benchmarks of the most time consuming subroutines parallelised with MPI together with scaling profiles indicate that using both MPI and OpenMP over the same regions will not give satisfactory scaling (table 1). There is not enough computation to compensate for the MPI communication costs in most subroutines. Currently, work is being done to increase parallel coverage as much as possible while keeping communication and synchronisation between MPI processes to a minimum.

Difficulties

During both stages of the parallelisation of TINKER, a number of issues that affect the performance and the effectiveness of the parallelisation have been observed:

- The fine-grained structure of the code, i.e. many small subroutines that are called many times, make it hard to parallelise;
- Memory issues: large memory usage and indirect memory addressing, produce bad memory locality and lead to memory trashing with a degradation in the overall performance;
- Large numbers of global variables, makes it hard to identify dependencies between different variables and subroutines.

References

[1] J. W. Ponder et al. , J Phys .Chem. B, 2010, 114(8), 2549-2564.
[2] J. W. Ponder and F M Richards *J. Comput. Chem.* 1987, 8, 1016-1024.
website : http://dasher.wustl.edu/tinker/
[3] D.A. Case, et al. (2014), AMBER 14, University of California, San Francisco.
website : http://ambermd.org
[4] O. Demerdash, E-H Yap and T. Head-Gordon, *Annu.Rev. Phys. Chem.* 2014(65): 149-74.
[5] http://ambermd.org/amber10.bench1.html#jac, last visited on 9th April 2015.
[6] G.M. Amdahl, *AFIPS Conf. Proc.*, 1967 (30): 483-485.