Barcelona Supercomputing Center
Centro Nacional de Supercomputación

TECHNICAL REPORT 2
BSC-INB
COMPUTATIONAL BIOLOGY PROGRAM

Jordi Camps (BSC-INB)

Molecular Dynamics simulations on Mare Nostrum

Jordi Camps. Barcelona Supercomputing Center-Centro Nacional de Supercomputación (Spain) jordi.camps@bsc.es

Abstract

This report presents a benchmark of three programs more commonly used for molecular dynamics simulations, with special emphasis on their behavior in parallel machines like Mare Nostrum.

Introduction

Molecular dynamics (MD) is a technique first developed in the middle seventies and applied to the protein world in the late nineties. The theoretical basis of MD is anchored in the basic rules of classical mechanics and the works of Newton and Langevin. The objective of MD is to reproduce a time-trajectory of the structure of a molecule, in our case protein. This is done by numerical integration of the equations of motion for a system whose interactions are defined by a simple atomic force-field like that shown in eqs 1-3. A standard MD step starts with the evaluation of the potential energy (eqs 1-3). Analytical derivation of the potential energy leads the forces acting on each atom, which using Newton's second law provides the acceleration. Integration of the accelerations yields new velocities, whose integration provides the new positions, completing then the integration step.

$$E = E_{bonded} + E_{non-bonded} \tag{1}$$

$$E_{bonded} = \sum_{bonds} K_s (l - l_0)^2 + \sum_{angles} K_s (\theta - \theta_0)^2 + \sum_{torsions} \sum_{i=1}^{3} \frac{V_i}{2} \left(1 + \cos(i\varphi - \xi)\right) \tag{2}$$

$$E_{non-bonded} = \sum_{a,b} \frac{Q_a Q_b}{r_{ab}} + \sum_{a,b} \left(\frac{C_{ab}}{r_{ab}}\right)^{12} - \left(\frac{D_{ab}}{r_{ab}}\right)^{6} \tag{3}$$

where l and θ stands here for bond lengths and angles, with subscript 0 for equilibrium values, K_s and K_b the associated force constants; Φ are torsion angles, V_i are the potentials associated with the Fourier terms used to represent torsions, ξ is the phase angle, Q are atomic charges, C and D denote the van der Waals parameters, and r_{ab} stands for interatomic distance.

For stability reasons the integration step needs to be shorter than the faster movement in the system, which forces us to use integration steps in the femtsecond range. This means that to collect 1 nanosecond of trajectory we need to perform 1 million integrations and a billion to collect a millisecond.

The cost of a MD simulation depends mostly on three considerations:

- **The size of the system:** larger the system greater the number of interactions and accordingly the cost of the simulation. Standard systems studied now are in the range of 10-20000 atoms. Some simulations have been reported for larger systems (around 100000 atoms) and a few groups have reported simulations for even bigger systems like membranes, ribosomes,…, approaching 1 million atoms (the counting considers always both macromolecule and solvent atoms).

- **The simulation conditions:** different technicalities have a direct influence in the cost of the calculation. The "rule of thumb" is that as we make simulation conditions closer to the biological environment calculation is more expensive. The current "state of the art" is to simulate the system under the constraints of constant pressure and temperature, using large boxes of solvent periodically reproduced in the space. Fast versions of Ewald's method are used to represent distant interactions out of the primary interaction cell. Integration steps of 1 fs (for some integrators 2 fs) are used.
- **Length of the trajectory:** First MD simulations were done in the picosecond time scale. Current "state of the art" for 10-20000 atoms is in the 10-50 ns range. A few simulations have been reported in the range of 100 ns and the microsecond time scale can be considered a "grand challenge" accessible only for a few groups in the world. To reach the 10 ns is a major effort involving years of CPU for larger systems (100000 to 1 million atoms). Considering that most biological processes happen in the millisecond time scale is clear that the MD community leads computers at least 3 orders of magnitude faster than current ones.

The development of MD codes started in the late seventies and many human-years of work are behind these codes, which in fact contain many other functions than the simple MD calculation. The first code developed was CHARMM, originated by the work of Karplus's group. Almost simultaneously was the development of GROMOS by van Gunsteren and coworkers. A few years later Kollman's group developed the first versions of AMBER and former van Gunsteren's students developed GROMACS, the first program developed thinking specifically on computer efficiency. More recently K.Schulten's group developed NAMD by rewriting from scratch existing codes searching in this case for better parallelism. All the groups developing codes also created their own force-field by fitting equations to experiments of high level quantum mechanical data. This is the case of CHARMM-force-field developed by Karplus's group, GROMOS-force-field by van Gunsteren's group and AMBER-force-field by Karplus's group. These and the OPLS force-field developed by Jorgensen and coworkers are considered the most powerful ones for the study of proteins.

In a recent work we performed a systematic comparison of force-fields in the study of equilibrium dynamics of proteins [16]. Here we became interested in the study of computational performance of three of the most used second-generation computer programs: GROMACS, NAMD and AMBER. In particular we explored the serial performance (single processor runs) and the parallel performance in Mare Nostrum computer.

Benchmarking the simulators

It is obvious that serial performance in this kind of programs is important, but parallel performance and scalability are the real crucial point when the program is going to be used in a massive parallel machine like Mare Nostrum.

In the Barcelona Supercomputing Center we have been working with three widely used simulation program suites. All of them can work in serial and parallel versions:

- Amber v8 [1]
- NAMD v2.6 [2]

- Gromacs v3.2.1 [3, 11]

To perform the benchmark we selected three proteins of different size with PDB codes 1CQY [4] (starch binding domain of bacillus cereus beta-amylase), 2HVM [5] (Hevamine A) and 1GND [6] (guanine nucleotide dissociation inhibitor, alpha-isoform).

The sizes of the systems (see Table 1) cover the range of small to large proteins:

Protein	**Protein residues**	**Protein atoms**	**Total atoms**
1CQY	99	1038	28310
2HVM	273	2701	31550
1GND	430	4342	66213

Table 1: Sizes of the systems used as input to the simulators that we are benchmarking

All trajectories were collected using “state of the art” simulation conditions using periodic boundary conditions with Particle Mesh Ewald technique to reproduce long range electrostatic effects and keeping constant pressure and temperature [16].

Serial performance

The first MD codes were developed with a serial approach because no parallel facilities were easily available. Some of them were later ported to a parallel environment, like Amber and Gromacs. NAMD was created having a parallel environment as the main scenario where it would execute.

Gromacs

The Gromacs program suite is a comprehensive collection of preparation, simulation and analysis programs. They follow the UNIX philosophy of many small and specialized utilities which achieve their objective when working together.

Gromacs was written using C [15], and the effort during development was in the speed and performance of the serial execution.

We can have the simulator compiled in four different ways depending upon it works with single precision floating point operations or double precision floating point operations and also if the code is compiled to work in serial or in parallel.

There have been important efforts to speed up the most costly calculus of the simulation, adapting the code specifically for different architectures. The inner loops have been heavily optimized, having assembly code for x86, Alpha and PowerPC platforms.

Working with single precision floating point operations, the code can be vectorized by the hardware of some current processors, speeding up some operations. But these loops are only available in single precision mode. When double precision floating point operations are performed, the data is too big to fit into the registers in a useful way and the vectorization is not available, thus offering a quite worst performance.

Amber

The Amber program suite comprises utility programs for the preparation of the input files, programs for the simulation, programs to analyze the results and also a program specially created to maximize the performance of the simulation when the simulation parameters meets certain requirements.

Pmemd (the new Amber simulator engine) has been written using Fortran-90 [12], while the original simulator (called sander) was written using an older version of the Fortran standard, Fortran-77 [13]. Pmemd was created to speed up MD simulations under Particle Mesh Ewald periodic boundary conditions which was prohibitively slow in the old sander module of AMBER.

Pmemd is a kind of sander rewriting from scratch. The new language allows the developers to use improved data structures and dynamic memory allocation, which makes the program lighter. The direct force evaluation code has been also rewritten, resulting in a code which uses considerably less memory. It has been programmed trying to repeatedly use the data which have some kind of locality, thus rising the cache hit ratio and therefore, reducing the time needed by a simulation.

NAMD

The NAMD program suite itself is the smallest suite of the three molecular dynamics suites that we are analyzing now. It contains only the main simulator and a few utilities to prepare the inputs for simulation.

The analysis and visualization of the results are done using other utilities from the same research group, like VMD and BioCoRE.

NAMD has been developed using C++ [14], but having a workstation's cluster as the natural execution environment of the simulator, hence a parallel communications library is used in the core and an appropriate object definition suitable for distributing the work across the processors. This architecture is also valid when we are using only one processor because the only difference is that the communications are local and do not have any latency, just the time to copy the data in-memory.

There are some approximations applied to speed up the algorithm:

- Use of three levels of integration loops:
    - Inner loop: bonded forces
    - Middle loop: Lennard-Jones and short-range electrostatic forces
    - Outer loop: long-range electrostatic forces

  The forces on the outer loop vary slower than the forces in the inner loop, so seems natural to compute the forces on the outer loop less frequently than the forces in the inner loop.

  This approach reduces the amount of operations needed along the simulation while maintaining a high level of accuracy.

- SPME: Smooth Particle Mesh Ewald. It is a refinement of the original PME method that uses B-splines to approximate the forces on the atoms. The continuity of B-splines functions and their derivatives makes the analytical expression of forces easy to obtain and reduces the number of FFTs by half compared to the original method.

Serial performance

Benchmarks for the serial version are shown in Figure 1. It must be noted that all the metrics obtained and used in this report involve only the total time used by the application since the beginning until the end of the program (also known as wall clock time). No conclusions about the efficiency of the codes can be derived from this data.

Amber-SANDER programs are the slowest in serial execution, with small gains with pmemd over sander, specially over bigger systems.

NAMD has similar timings to the Amber-SANDER ones. It is faster than pmemd in all cases but its performance is not extremely good.

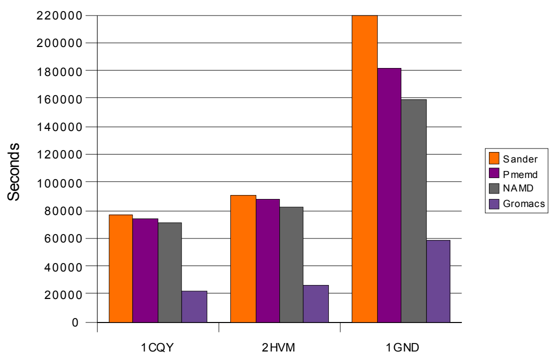


Figure 1: Timings of the different codes running on one processor

Gromacs is the clear winner of the benchmark. Its timings are two or three times faster than its competitors. This fact is probably due to the different goals that the developers had in mind. Amber and NAMD developers tried to make their code parallel and scalable, paying special attention to the parallelization methods and communications overhead. On the other hand, Gromacs developers has focused on implementing the serial routines in the quickest way, devoting so many time to the optimization of the internal loops, where the code spends most of its time. They do not only optimized their code in an algorithmical way, they get down to the assembly level in order to codify specialized loops for every architecture, getting an instruction pipeline with a very low number of stalls and idle cycles.

Code parallelization

Serial code is the basis for the posterior parallelization. A given serial code can be fast or slow, and the parallelization difficulty usually changes with the code efficiency: a faster code means a greater synchronization/computing ratio.

The different programs have chosen different parallelization strategies which state clearly that there is no consensus on the best methodology to use in order to partition the same problem in calculable partitions.

Now I will explain how the different simulators addressed this problem and which have been the problems that aroused with each decision.

Gromacs

Gromacs was created having low-cost computers in mind. This means that they do not rely on having neither fast networks nor full interconnection schemes between the processors. The basis for its intercommunication paradigm was the assumption of a ring topology [10]. They choose this topology because a ring can always be mapped onto a more general interconnection scheme like a hypercube or a tree, and in the case we do not have expensive hardware, this topology can also be used.

Because of each element of the ring has only two neighbors, left and right, it was decided that the better decomposition that can be used was to decompose the simulation cell along the X axis, slicing it and assigning each slice to a processor. This kind of decomposition has advantages over particle decomposition, like the easiness of load balancing, which is very useful in order to obtain a good scaling factor. But one of the disadvantages of this approach is that the parallelism depends on the input shape.

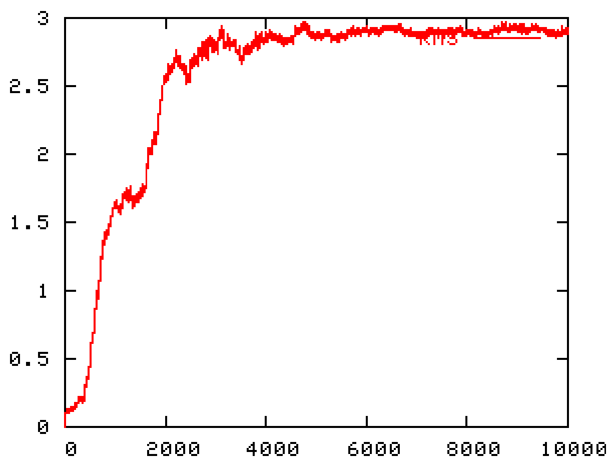


Figure 2: RMSd plot (nm vs. ps) of a simulation made with Gromacs applying the G43a1 force field.

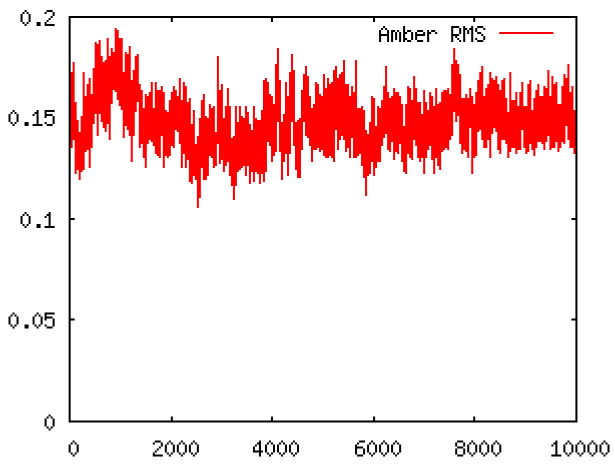


Figure 3: RMSd plot (nm vs. ps) of a simulation made with Amber applying the Amber force field.

This data distribution scheme needs every processor being aware of the coordinates for all the particles all the time while distributing the forces back to the other processors. The velocities are kept on the "home processor" of each particle, and the fixed interaction lists (for bonded interactions, angles, dihedrals…) are also kept on a single processor, provided that care is taken that every processor gets a similar amount of bonded interactions.

Another parallelization used in some parts of the code is the one related to the FFT calculus. It is performed through the FFTW library, which uses MPI to parallelize the calculus.

This decision is against the original idea of permitting only have a physical ring topology because the FFTW library is not restricted to use only a ring topology. In that case, the underlying MPI implementation will have to take care of the messages addressed to processors which are not the neighbors of the sending processor in a ring topology, having an important performance impact and increased network usage.

We ran some tests with this simulator, identifying some incoherent results compared with other simulations. This can be seen in Figure 2 and 3, which show artifactual trajectories in GROMACS due to a sharp transition to unfolded structures of proteins after 1 nanosecond (note the large jump in root mean square deviation in Fig 1), when the protein is found stable with other programs (see AMBER profile in Fig. 2). Analysis of the trajectories discarded the existence of force-field artefacts, since while OPLS-simulations performed in NAMD behave well, those performed in GROMACS unfold the structure. The error reported in Figure 2 was found for all the proteins in our MICROMODEL database (see data at http://www.bsc.es/). The time of the trajectory where corruption appears is stochastic (the same simulation performed twice is corrupted at different points) and depends on the number of processors: larger the number sooner the corruption.

These results lead us towards an issue with the parallelization code, but at this point the reasons for the stochastic behaviour of parallel versions of GROMACS in Mare Nostrum are unclear, since recompilation was carefully done and all tests and benchmarks were successfully passed. In any case MN users are strongly encouraged to check carefully trajectories collected with parallel version of GROMACS 3.2.1.

Amber

There is a little more information referring to the parallelization methodology used in sander (the main simulator) and pmemd (the specialized simulator) than to the serial code.

Amber was first parallelized for the hardware found in CRAY T3D and T3E. Extensions to support other architectures were added later, including a general Message Passing implementation.

Parallelization in Amber works using a replicated data approach [7]. This is a flexible method, but suffers from communications overhead when collecting and broadcasting coordinates and forces for the entire system to all processors at each step.

Because most of the time is spent in the non bonded interactions and the Particle Mesh Ewald (PME) algorithm, hence the main effort has been directed to this area.

In the Ewald method, the sum of Coulomb’s Law terms is expanded into several sums:

$$E(electrostatic)=E(direct)+E(reciprocal)+E(correction)$$

$E(direct)$ is identical to the sum used with the cutoff method. $E(reciprocal)$ depends on a discrete Fourier transform and $E(correction)$ serves the purpose of eliminating the terms that has been double-summed in the reciprocal calculus.

The direct sum can be computed in each processor once the atom pair list is generated. The simulation box is divided in subcells so that their dimension is at least 1/3 of the cutoff distance. Although the search part of the list building scales perfectly, the setup of the subcells does not. Each processor executes the direct sum on its portion of the pair list and stores the resulting energy and forces locally. Global summing is done after all other contributions. The reciprocal sum needs much more effort. The entire reciprocal energy and force calculation is distributed by dividing the charge grid into equal parts for each processor. The constraint was to limit the breakdown of the grid to

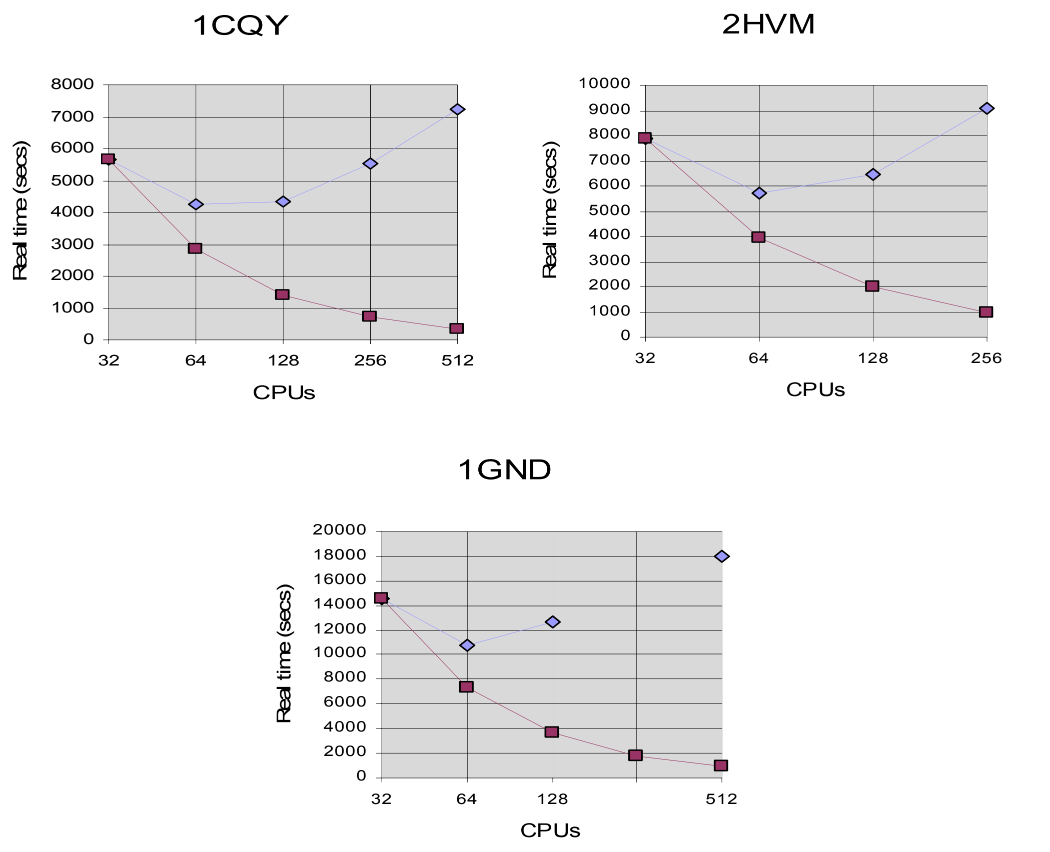
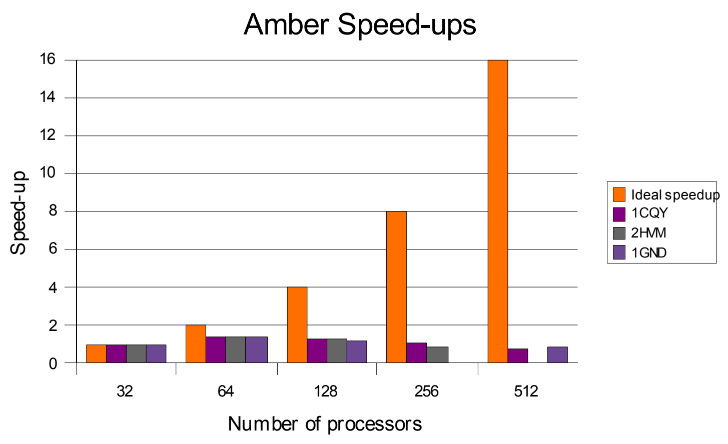


Figure 4: Timings for the simulation of the different solvated proteins with Amber. It is clear that the program scale until 64 processors only. The ideal scaling line is also drawn.

xy planes. This limits the scaling to the number of planes. If we have a simulation box of dimensions 60x60x60Å and a charge grid of the 60x60x60, the space is divided in 60 planes, and using more than 60 processors give no speed up because the extra processors above 60 remain idle. The best performance is obtained when using 20 or 30 processors, because using 60 processors needs to communicate much messages and the speed-up is low.



This plane division is needed to perform a 3D FFT on it. This 3D FFT is accomplished by performing 1D FFTs in each of the three Cartesian directions. The 1D FFTs in the *x* and *y* directions can be performed locally, but for the *z* direction, the grid must be transposed across the processors, and this transposition is the only work that will not scale in a desirable way with increasing processors.

Further scaling can be achieved by breaking planes in half and assigning adjacent half planes to adjacent processors.

Another optimization in the calculus when we have a large number of processors is the concurrent calculus of the direct and reciprocal sum. Because these two sums are independent, we can split the processors in two groups, one for the computing of the direct sum and another for the execution of the reciprocal sum. The partition sized can be determined by trial and error using short test runs, and when the division is determined further redistributions of processors will not be necessary. This parallelism method is also advantageous when we have a high latency network because we reduce the number of processors that must communicate between themselves. Although the total data sent is almost the same, the number of messages is considerably smaller, thus reducing the time used waiting for data to arrive.

The benchmarks for Amber give the timings shown in Figure 4. The speedups can be seen in Figure 5.

NAMD

NAMD team is proud to announce that their code scales to hundreds of processors on high-end parallel platforms, and the benchmarks executed gives a quite good performance [8].

As said in the serial description, NAMD’s source code is written in C++ and uses the Charm++ library for the parallelization. Charm++ is written on top of MPI, which makes it portable across a wide range of computers. Parallelization in NAMD code is achieved by distributing the atoms across the processors as evenly as possible [9].

NAMD uses a way of decomposition that easily generates the large amount of parallelism needed to occupy thousands of processors. Charm++ parallel objects and data-driven execution adaptively overlaps communication and computation, hiding communication latencies. In Charm++, objects may migrate between processors at runtime. This migration is controlled by the Charm++ load balancer.

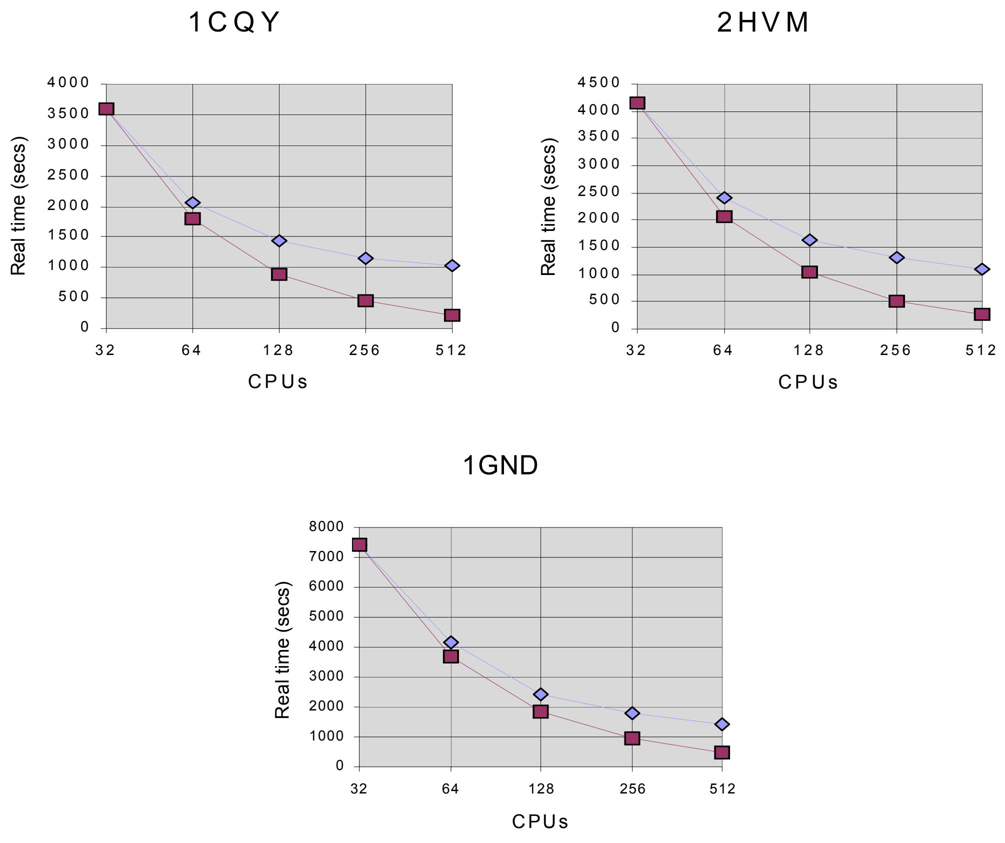


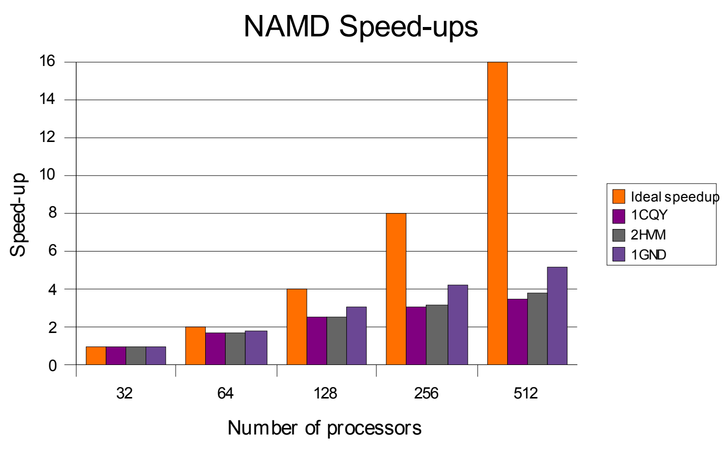
Figure 6: Timings for the simulation of the different solvated proteins with NAMD. It is clear that the program scale quite well. The ideal scaling line is also drawn.

The simulation cell is divided in a three-dimensional grid, having each grid element the cutoff size necessary that only the 26 nearest grid elements are needed to evaluate the bonded, van der Waals and short-range electrostatic interactions. Once we have the simulation cell divided, we distribute each grid element to a processor, along with a (roughly 14 times) larger number of compute objects (one for each pair of neighboring grid elements) for the calculus, grouping compute objects responsible for the same grid element together on the same processor. The hydrogen atoms are always assigned to the same grid element as the atom they are related to. When there are more grid elements than processors, which is the common case, nearby grid elements are assigned to the same processor in order to avoid unnecessary network communication.

But in the case that we have more processors than grid elements, we can generate more parallelism through options that double the number of patches in one or more dimensions.

The atom-grid element mapping is re-evaluated at regular intervals to adapt to the new shape of the molecules.

When the simulation begins, grid elements are distributed according to a recursive bisection scheme. After the simulation run for several time-steps the program trigger the initial load balancing which is the most aggressive. After this initial load balancing only small refinements are made. Two more cycles of load balancing follow immediately, after which load balancing occurs periodically to maintain load balance.



Benchmarks executed by NAMD authors suggests that the limits of NAMD's parallel scalability are mainly determined by atom count, with one processor per 1000 atoms being a conservative estimate for good efficiency. But it is obvious than with a high number of atoms, the scaling will be much better because each grid element will be able to compute more with less communication, being this estimation a lower limit in the atom number to obtain good efficiency.

Also, increased cutoff distances can have an effect on scaling as the number of grid elements is smaller.

PME is the most time-consuming calculus in the simulation with NAMD, and the reciprocal sum is the main point, like in the other programs. The reciprocal sum is currently parallelized only to the size of the PME grid. But the interleaving of computation and communication provided by the Charm++ framework allows NAMD to compute the direct sum while waiting for the data needed to compute the reciprocal sum, effectively hiding the network latency.

The benchmarks for NAMD give the timings shown in Figure 6.

Benchmark results

Given the benchmarks results, it seems quite clear that NAMD is one of the codes that scale better and scaling to up to 1000 processors will be tested in future benchmarks for systems containing more than 100000 atoms. Amber shows an important performance loss when scaling to more than 64 processors, mainly due to the work distribution in the reciprocal sum of the PME calculation. As explained before, using more processors than the number of planes in which the simulation cell is sliced gives no speed up and in fact leads us to absolute increases in the CPU time (see Figure 4).

NAMD can keep up its performance because the high number of work units that are generated for each simulation cell, so a better load balancing can be performed, alleviating the problem of having not-working processors waiting for the others to finish.

Finally, Gromacs has not been tested, but taking into account the method used to distribute data across the processors, which is quite similar to the method used by Amber, I will bet for an Amber-like behavior. Gromacs is obviously limited by the spatial decomposition, and if there are more processors than work slices, they will not be able to do any work and will remain idle.

Despite this results, more analysis are being carried on to determine the efficiency of the different programs. Is it clear how the programs behave out-of-the-box, but it is not clear if we can get better results in Mare Nostrum or if we are working at full speed.

Suggestions for development

The performance of "grand challenges" simulations require of very fast and parallel codes. Pmemd behaves well for a small number of processors and is then useful for massive projects like Model but not for single very long runs due to its very poor performance for more than 64 processors. NAMD scales very well, but at the expense of a smaller single-processor performance, and of the smaller stability of the integrator (often shorter integration steps than Pmemd need to be used). An additional shortcoming of NAMD is the scarce possibilities of the code when a non-standard simulation is planned (for example the code is not ready to perform free energy calculations other than those derived from steered MD simulations. GROMACS has the best per-processor performance, but its behavior in heavily parallel machines like MN is not recommended due to the existence of spurious errors. It is suggested that effort should be made in tracing the executions of NAMD and Amber to get efficiency data and improvement suggestions. With data from traced executions we can decide if we must try to improve the serial execution time of NAMD or try to improve the Amber scalability.

References

[1] Pearlman, D. A., Case, D. A., Caldwell, J. W., Ross, W. S., Cheatham III, T. E., BeBolt, S., Ferguson, D., Seibel, G., and Kollman, P. (1995). AMBER, A package of Computer Programs for Applying Molecular Mechanics, Normal Mode Analysis, Molecular Dynamics and Free Energy Calculations to Simulate the Structural and Energetic Properties of Molecules, *Computer Physics Communications*, 91, 1, 3:1-41.

[2] Kalé, L.; Skeel, R.; Bhandarkar, M.; Brunner, R.; Gursoy, A.; Krawetz, N.; Phillips, J.; Shinozaki, A.; Varadarajan, K.; Schulten, K.; (1999). *J. Comp. Phys.*, 151, 283.

[3] E. Lindahl and B. Hess and D. van der Spoel. GROMACS 3.0: A package for molecular simulation and trajectory analysis. *J. Mol. Mod.* 7 (2001) pp. 306-317

[4] Yoon, H.J., Hirata, A., Adachi, M., Sekine, A., Utsumi, S., Mikami, B. Structure of Separated Starch-Binding Domain of Bacillus cereus B-amylase. To be published

[5] Terwisscha van Scheltinga, A.C., Hennig, M., Dijkstra, B.W. The 1.8 Å resolution structure of hevamine, a plant chitinase/lysozyme, and analysis of the conserved sequence and structure motifs of glycosyl hydrolase family 18. *J. Mol. Biol.* v262 pp. 243-257, 1996

[6] Schalk, I., Zeng, K., Wu, S.K., Stura, E.A., Matteson, J., Huang, M., Tandon, A., Wilson, I.A., Balch, W.E. Structure and mutational analysis of Rab GDP-dissociation inhibitor. *Nature* v381 pp.42-48, 1996

[7] Crowley, M.; Darden, T.; Cheatham III, T.; Deerfield II, D. (1997). Adventures in Improving the Scaling and Accuracy of a Parallel Molecular Dynamics Program. *The Journal of Supercomputing*. Vol. 11, number 3, pp. 255-278

[8] M. Bhandarkar, R. Brunner, C. Chipot, A. Dalke, S. Dixit, P. Grayson, J. Gullingsrud, A. Gursoy, D. Hardy, J. Hénin, W. Humphrey, D. Hurwitz, N. Krawetz, S. Kumar, M. Nelson, J. Phillips, A. Shinozaki, G. Zheng, F. Zhu. (2006) NAMD User's Guide, Version 2.6

[9] Phillips JC, Braun R, Wang W, Gumbart J, Tajkhorshid E, Villa E, Chipot C, Skeel RD, Kale L, Schulten K. Scalable Molecular Dynamics with NAMD. J. Comp. Chem. 2005 Dec; 26(16):1781-802

[10] van der Spoel, D.; Lindahl, E.; Hess, B. (2006) Gromacs User Manual, Version 3.3

[11] H. J. C. Berendsen, D. van der Spoel and R. van Drunen. GROMACS: A message-passing parallel molecular dynamics implementation. *Comp. Phys. Comm.* 91 (1995) pp. 43-56

[12] Programming Language - Fortran - Extended (formerly ANSI X3.198-1992 (R1997))

[13] Programming Language FORTRAN (ANSI X3.9-1978). American National Standard, also known as ISO 1539-1980

[14] Programming Language C++. ISO/IEC 14882:1998

[15] Programming Language C - ISO/IEC 9899:1990 Information technology

[16] Rueda, M.; Ferrer-Costa, C.; Meyer, T.; Pérez, A.; Camps, J.; Hospital, A.; Gelpí, J. and Orozco, M. *A consensus view of protein dynamics*. PNAS (2007). vol. 104. no 3. (796-801)

Acknowledgements

The author thanks the help of Prof. M.Orozco and MMB group in the preparation and analysis of trajectories. The author also thanks the help of Dr. S.Girona and Mare Nostrum Support team in running the benchmarks. Finally the author is indebted to Profs. J.M.Cela, J.Labarta and S.Girona for help in the analysis of the benchmarks.