Large biomolecular simulation on HPC platforms II. DL_POLY, Gromacs, LAMMPS and NAMD

Hannes H. Loeffler,[a] Martyn D. Winn[a]

[a]*Computational Biology Group, Computational Science and Engineering Department, STFC Daresbury Laboratory, Daresbury, Warrington WA4 4AD, United Kingdom.*

Abstract

We have carried out a performance assessment of the four molecular dynamics (MD) packages NAMD, LAMMPS, Gromacs and DL_POLY. For that purpose benchmarks were run on the HPC platforms BlueGene/P, on HECToR phases 2a and 2b, and on a test cluster equipped with graphical processing units (GPUs). The dimeric ectodomain plus transmembrane helices of the Epidermal Growth Factor Receptor (EGFR) was chosen as a relevant test system of biological interest. The receptor was placed on a model membrane to create a system of about 465 000 atoms in explicit solvent. To generate a large size system the receptor–membrane assembly was replicated resulting in a total number of about 3 million atoms. Performance results are reported and compared to our previous numbers (Technical Report DL-TR-2009-002). We discuss the maximum number of nano seconds per day that can be achieved and give general and practical advice for MD simulations of large scale biological systems. In particular, we provide benchmark data for HECToR phase 2b with the new Gemini communication interconnect.

1 Introduction

This is the second instalment in a series of benchmark studies of popular molecular dynamics (MD) codes of biological interest on various hardware platforms including the UK's current national flagship HECToR. In the previous report (1) we presented performance results of AMBER (2), Gromacs

Email address: Hannes.Loeffler@stfc.ac.uk (Hannes H. Loeffler,).

(3) and NAMD (4). Since then updated software versions for Gromacs and NAMD have been released and new benchmarks are included in this report. We have updated our list of codes to include LAMMPS (5; 6) and DL_POLY (7; 8). Some of the hardware, like HPCx and HP Cluster Platform 4000, have been decommissioned already or will be retired soon. Hence, we will concentrate here on the BlueGene/P and Cray XT5h (HECToR phase 2a, only XT4 subsystem has been benchmarked) and Cray XT6 (HECToR phase 2b). Some early benchmarking results on a test cluster equipped with graphical processing units (GPUs) will be presented as well.

In the past few years, GPUs have emerged as viable platforms for compute-intensive scientific applications. Adoption of GPUs has been encouraged by the development of high level GPU programming languages such as CUDA and OpenCL. There have been a number of MD programs written for or extended to GPU systems, e.g. HOOMD (9) and ACEMD (10). Gromacs-OpenMM is a modified version of Gromacs (with limited functionality) which uses OpenMM to perform calculations instead of its own computational routines. OpenMM in turn has been ported to GPUs using CUDA. In this report, we focus on the CUDA port of NAMD (4; 11).

In the previous study (1), we have used a small test case of about 61 000 atoms and a larger one of about 465 000 atoms (465K system). Here, we wish to concentrate on larger scale systems and present benchmarks for the 465K system and a new system of nearly 3 million atoms (3M system). The 465K system is the hydrated dimeric ectodomain plus transmembrane helices of the Epidermal Growth Factor Receptor (EGFR), including up to residue 661 (numbering of the mature receptor) and liganded with the epidermal growth factor EGF, placed on a model 1–palmitoyl–2–oleyl–*sn*–glycero–3–phosphocholine (PamOleGro–P–Cho, POPC) bilayer. The 3M system is the tetramer from a previous study (12) replicated two times and placed on a large $450\times480\,\text{Å}^2$ POPC membrane to obtain a model of a large scale assembly of receptors on a membrane. All systems were run fully atomistically.

This report is provided in the hope that it will be useful as a general information resource for running biomolecular MD simulations as efficiently as possible. The results will give a first impression of the performance of the MD codes on HPC platforms and will therefore be useful to those seeking for comparative benchmarks. Also, and especially for first time users, the performance data will help to keep one's own benchmarking to a minimum. Performance assessment is ideally carried out during the setup procedure and/or equilibration phase of an MD simulation. Despite the extensive benchmarks presented here it is still advisable to carry out some performance studies to optimise runtimes as every individual case may be different and runtimes will also vary depending on the details of a particular hardware platform. Here, we summarise only results for standard MD simulation. More elaborate techniques will not necessarily scale in the same way.

It is also worth mentioning that the choice of a particular MD code will not only depend on its performance characteristics but also on the required features. Most MD programs have come quite a long way in supporting multiple force fields. The CHARMM, AMBER, and OPLS forces fields are typically fully supported in the most recent versions. On the supported feature side, however, the codes still vary quite markedly. Some features may be unique to a particular MD program or better supported.

2 Running the tests

The serial part inherent to all MD programmes (reading input files, setting up grids, etc.) has not been excluded from the Gromacs benchmark runs as that MD program reports only total run times. We have found that this part usually makes up only a few percent. Furthermore, determination would require additional runs to be carried out, e.g. by running a longer simulation and obtaining the serial part by subtraction from a shorter run. It also has been shown very recently (13) that the number of steps carried out for the system sizes in our study is sufficient to reproduce the scaling behaviour of long time production runs.

Most benchmark data reported here were determined from single runs. Tests have shown that runtimes typically do not vary more than five percent. Running multiple simulations, however, is very time consuming and also very expensive on external hardware (most notably runs on HECToR phase 2b with fewer cores per node than the maximum of 24, for discussion see below). The main emphasis of this report is to give a useful impression of scaling behaviour. Elaborate statistics would alter this only in an insignificant way. The largest impact would be found in the “flat” parts of performance curves, effectively meaning at high core counts around peak performance and beyond. For production runs, however, this is a region which should not be used as performance gains are either miniscule or performance is actually dropping (see the Results section below).

One note on the nomenclature applied here. We will use the term “core” to mean the physically smallest unit of processing. Since all MD programs in our study are MPI applications (with the exception of some NAMD binaries on BlueGene/P which bypass MPI for performance reasons) this means one process per physical core. On all platforms individual processors (CPUs) are made up of multiple cores (see Hardware subsection below). Nodes consists of one processor in the case of the BlueGene/P and the Cray XT4, and two processors in the case of the Cray XT6.

2.1 Test systems and force fields

The Epidermal Growth Factor Receptor (EGFR) dimer ectodomain was modelled and attached to two transmembrane (TM) helices as described previously in (12). The initial coordinates were taken from this study (ca. 840 000 atoms) by splitting the tetramer system into half. The EGFR dimer lies flat on a 1–palmitoyl–2–oleyl–*sn*–glycero–3–phosphocholine (POPC) model bilayer and was immersed in water. Sodium and chloride ions were added and their concentration adjusted to about $0.15\,\mathrm{mol\,l^{-1}}$. This procedure resulted in a system of about 465 000 atoms (465K system).

To obtain a larger scale system but still one of biological relevance we replicated the tetramer (12) two times and placed the protein on a large $450\times480\,\text{Å}^2$ POPC membrane. Thereby we arrived at a system size of nearly 3 million atoms (3M atoms) as a model for a population of receptor multimers. Water molecules and ions were added as in the smaller system.

The force fields applied with NAMD, LAMMPS, and DL_POLY were the all-atom CHARMM27 force fields for proteins (14; 15), lipids (16; 17), ions (18) and TIP3P water (19; 14). Since full support for the CHARMM force fields has been integrated into Gromacs only recently (20), we used the Gromos force field for the protein and a mixture of OPLS parameters and the Berger set (21) for the lipids. For water the SPC model (22) was chosen. As the force fields for both protein and lipids were united-atom we added additional water molecules and ions to match the atom numbers of the all-atom force field setup.

2.2 Simulation parameters

Comparing MD simulation packages is complicated by the fact that each program supports certain features that may not be available in another. Typically, time step integrators or temperature and pressure control are implemented with different algorithms or techniques. Overall, however, we do not expect a large impact on runtimes varying these parameters. Where possible we have tried to use as compatible runtime parameters as possible. See the Appendix for a summary of input files used.

The most time consuming part of an MD simulation algorithm is the computation of the non-bonded interactions and here most notably the electrostatic interactions. Most packages use a variant of the Particle Mesh Ewald (PME) method (23) but LAMMPS makes use of the Particle-Particle Particle-Mesh (PPPM) algorithm (24). We mostly went with the default PME and PPPM parameters except for the convergence criterion and the grid size which has to be set by hand in NAMD. Whereas NAMD does load balancing steps in the initial phase and also periodically at later steps the number of nodes assigned to PME calculations has to be adjusted manually in Gromacs. A utility helping in determining the optimal number of nodes automatically, `g_tune_pme`, has not yet been used by us but it was typically found that about 25% percent of total nodes give the best performance (3; 13). We expect that tweaking the PME/PPPM parameters or in general any parameter that affects load balancing (NAMD allows that) may lead to an increased performance. Of course, the trade-offs due to possibly reduced accuracies must be evaluated.

For all benchmark runs 10 000 MD steps were carried out except in the case of DL_POLY where only 5 000 steps were run. This is a number that still conveniently accommodates runs on low core counts and time-limited batch systems but is also sufficiently high to yield reliable benchmark data as outlined above (13). Processor counts have been varied in as wide a range as possible but the large 3M system could only be run with higher core numbers. It is also worth mentioning that NAMD needs at least 2 GB of memory for systems beyond 2 million atoms, a number that increases with increasing core numbers due to the way load balancing is implemented (25). This is a particular problem on the BlueGene/P which has only 2 GB of memory per node. All runs for the 3M system had to be carried out therefore with 1 process/node despite the BlueGene/P allowing for up to 4 processes/nodes. In all other cases the number of processes per node has been kept at the maximum possible although below we also present data with varying process numbers per node on the Cray XT6 to evaluate their impact on performance.

The topology files have been created as follows. For NAMD we used `psfgen` through the graphical user interface `vmd` (26). LAMMPS comes with a Perl script called `charmm2lammps.pl` (version 1.8.1 in our case) that aids in converting CHARMM's PSF format to LAMMP's format. We directly used the PSF and PDB files generated from `psfgen`. For DL_POLY the new DL_FIELD (27) program was used to create the respective input files from the PDB file. For Gromacs we used the Gromacs utility `pdb2gmx` to generate the topology files for the protein. In addition, we download the input topology file for POPC from P. Tieleman's web page (`http://moose.bio.ucalgary.ca/`).

2.3 Software

An overview of the four MD software packages used in this study is given in Table 1. We list the version numbers, the force fields used and whether the programs had been compiled by ourselves or by someone else. Executables on Hector have been prepared by the support team. NAMD executables for the BlueGene systems have been compiled by the authors of the software but

Table 1
Overview of the four MD simulation packages benchmarked in this report.

Program	DL_POLY	Gromacs	LAMMPS	NAMD
Versions	4.00	4.0.7	29/5/2010	2.6
		4.5.3	04/7/2010	2.7 [a]
Force field	CHARMM	Gromos/Berger	CHARMM	CHARMM
Compiled HECToR [b]	—	pre	pre	pre
Compiled BG/P	pre	self	self	pre/self

[a] Final 2.7 and beta versions 1, 2, or 3 depending on availability.
[b] Abbreviations used: pre is pre-compiled and self means compiled by the authors of this report.

the new version 2.7 was compiled by us, as was the LAMMPS and Gromacs software.

NAMD was available in pre-compiled form on all platforms for the older version 2.6. On some hardware we only had the chance to benchmarks various beta versions of 2.7 due to availability.

GPU support using the CUDA toolkit is included in the source code distribution of NAMD. CUDA code is used to evaluate non-bonded forces (11). The associated CPU is used for forces between bonded atoms, computing energies and load balancing calculations. Version 2.7b3 of NAMD was compiled from source on the cseht cluster (see below) using the Intel v11.0 compilers, following the standard procedure documented on the NAMD download page. As instructed there, the Charm 6.2.1 library was built without CUDA support. Jobs are launched with charmrun with the argument +p specifying the total number of GPU processes ($N \times PPN$). The NAMD load balancer works at the level of these GPU processes, while CUDA itself handles distribution amongst the streaming multiprocessors within each GPU.

2.4 Hardware

An overview of the hardware used is given in Table 2. The detailed architecture of the systems is more complicated than presented, in particular the communication networks as well as data I/O and storage are all crucially important for performance. However, the simplified hardware data will be sufficient to get an understanding of the benchmark results below.

The GPU tests were performed on the STFC Daresbury cseht cluster facility which includes 8 Intel Nehalem servers, each of which is connected to an NVIDIA Tesla S1070 server containing 4 Tesla T10 GPUs. Each Nehalem node

Table 2
Overview of the HPC hardware platforms used for this report.

Machine	Cray XT5h [a]	Cray XT6/XE6 [b]	BlueGene/P
proc type	AMD Opteron	AMD Opteron	IBM PPC 450
proc clock rate	2.3 GHz	2.1 GHz	850 MHz
cache	L3 shared	L3 shared	
proc/node	quad-core	4×6-core [c]	quad-core
total# cores	12 288	44 544	4 096
memory/node	8 GB	32 GB	2 GB

[a] HECToR phase 2a. Only Cray XT4 subsystem benchmarked.
[b] HECToR phase 2b. The XE6 system is the XT6 upgraded with the Gemini communication interconnect in replacement of the SeaStar2.
[c] Packaged as 2 Opteron processors.

can thus support up to 4 GPU processes. Jobs are reported according to the number of nodes ($N \leq 8$), and the number of processes per node ($PPN \leq 4$). Each GPU contains 240 processor cores arranged into 30 streaming multiprocessors (SM). Each SM thus contains 8 processor cores, as well as 16 kB of fast locally shared memory. Threads are synchronised within each SM, so that it acts as a SIMD unit. Each GPU has 4 GB of device memory.

3 Results

3.1 New versions of NAMD and Gromacs

The new versions NAMD 2.7 and Gromacs 4.5.3 have recently been released. We will therefore start with comparing performance with the predecessor versions. Some of the results presented below have been benchmarked with various beta versions of NAMD 2.7 but we have not found noticeable differences to the final release.

Figure 1 (left) depicts the comparison between NAMD 2.6 and NAMD 2.7b2. The newer version is about 10 to 25% faster than the older one as the inset shows. The relative speedup varies quite a bit but it must be mentioned that all data points have been calculated from a single benchmark run. NAMD's 2.7b2 performance appears to peak earlier at 2048 cores but is faster by about 25% than NAMD 2.6. The test runs have been carried out on the Cray XT4 with the 465K system.

The performance improvements of Gromacs 4.5.3 over 4.0.7 are shown in Fig-

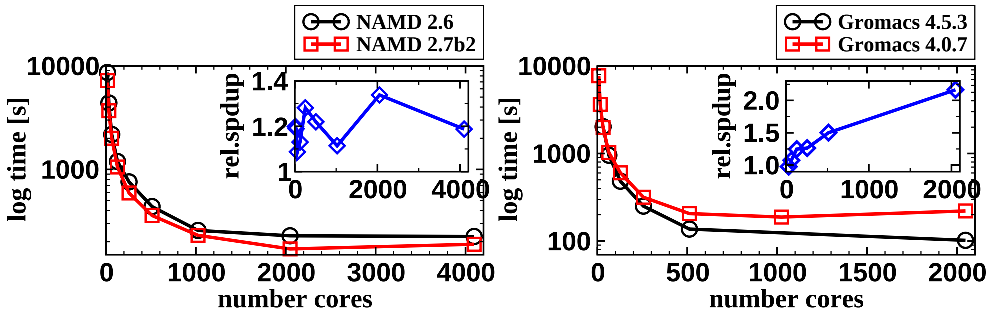

Fig. 1. Performance comparison between NAMD 2.6 (black) and NAMD 2.7b2 (red) as obtained from the Cray XT4 with the 465K system. The inset shows the relative speedup of the newer version relative to the older one. Peak performance occurs earlier with NAMD 2.7b2 at 2048 cores but requiring less compute time. The equivalent results for Gromacs 4.0.7 (red) vs. 4.5.3 (black) are shown in the right hand panel. Please note the logarithmic scale of the time axis.

ure 1 (right). The newer version displays considerably lower runtimes at higher core counts. The relative speedup was highest at 2048 cores by a factor of over 2.

In contrast, on the BlueGene/P we find very little difference in the performance of the two NAMD versions. The program can be compiled with MPI support or a version that bypasses MPI for increased performance. We find that the MPI version of 2.6 (downloaded from the NAMD web page) and 2.7 (compiled by us) show essentially the same runtimes. The non-MPI 2.7 version (also compiled by us) is up to 10% faster at higher core numbers but less so at lower core numbers. In the data below we will only present results from NAMD 2.7 non-MPI for the BlueGene/P except in the case of the 3M system where the NAMD 2.6 MPI version was used due to memory problems.

3.2 Optimised routines in LAMMPS

LAMMPS has been in development for many years by now but has gone through several changes in the programming language used from Fortran 77 (LAMMPS 99) to Fortran 90 (LAMMPS 2001) to currently C++. As switching the language essentially means rewriting the software it is likely that further development will have an impact on the performance. At the moment the source code comes with a set of separate optimised routines for the computation of the long range forces. Figure 2 shows the performance improvements in these routines in comparison with the standard routines.

The optimised version is up to a third faster than the non-optimised code. A minimum is not found in the speedup curve, i.e. LAMMPS scales to the highest core count of 4096 on the BlueGene/P. Further benchmarks were run

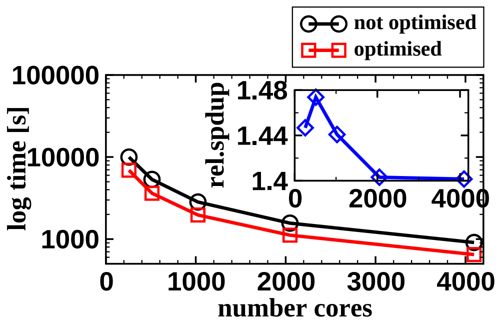

Fig. 2. Performance comparison between LAMMPS not optimised (black) and LAMMPS optimised (red) as obtained from the BlueGene/P with the 465K system. The inset shows the relative speedup of the optimised version relative to the non-optimised one. The code scales to the highest core counts in both cases. Please note the logarithmic scale of the time axis.

with the optimised routines on the BlueGene/P but not the the two Cray machines due to unavailability.

3.3 Benchmarking the 465K system

Benchmark results for the 465K system run on the BlueGene/P and both the Cray XT4 and XT6 are shown in Figure 3. All benchmarks have been run fully populated which means that all cores of a node have been utilised: 4 per node on the BlueGene/P and Cray XT4, and 24 on the Cray XT6.

The benchmarks on the BlueGene/P demonstrate that Gromacs 4.5.3 (floating point version) performs fastest under all core counts, i.e. this MD program shows the highest absolute performance. NAMD is the second fastest program but scales less well than Gromacs as evident from the less steep slope. While with 256 cores LAMMPS is about five times slower than NAMD the performance gap is steadily decreasing with increasing core numbers reaching a factor of about 2.5 at 4096 cores. DL_POLY performs faster than LAMMPS in absolute terms up to about 1000 cores but runs slightly slower on higher core counts. Also, DL_POLY scales less well than the other codes on the BlueGene/P. All programs scale to at least 2048 cores.

On the Cray XT4 we see a similar picture between NAMD 2.7b3 and LAMMPS (version from May 29, 2010). NAMD performs faster on an absolute scale but LAMMPS scales less well with increasing numbers of cores. The performance of LAMMPS is about 3 times slower with 64 cores but only about 2 with 4096 cores just as with core numbers of 32 and below. We also show benchmarks with Gromacs in double and single precision in Figure 3 (bottom left). While all other MD programs presented here work in double precision Gromacs can also be compiled in single precision. It has been shown recently for both Gro-

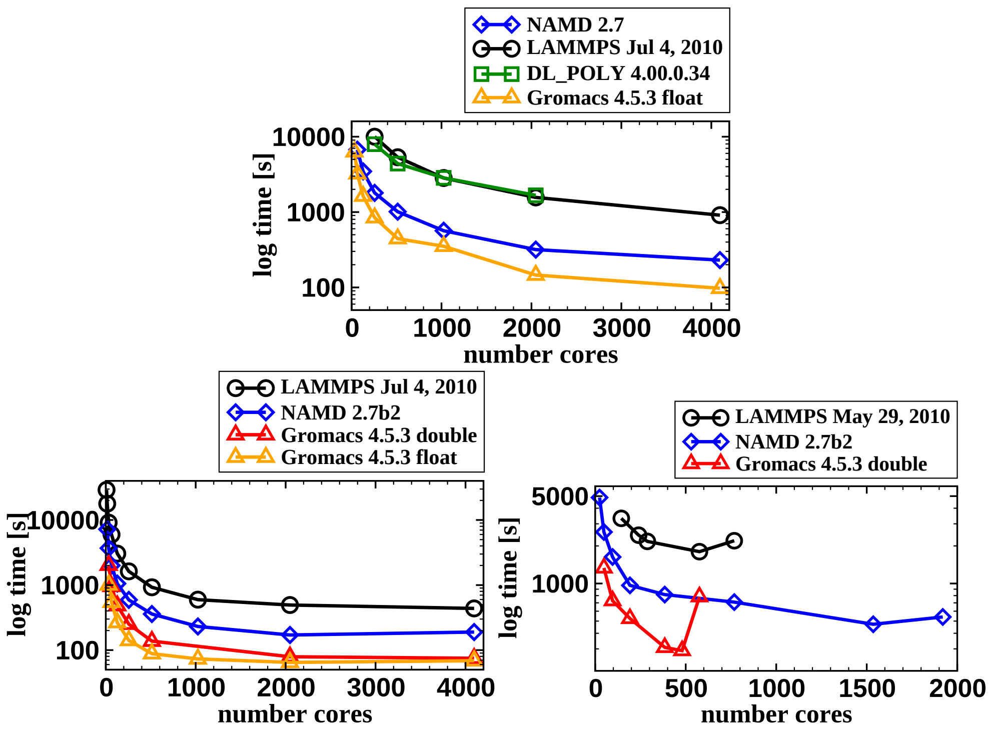

Fig. 3. Comparative benchmarks on the BlueGene/P (top), Cray XT4 (bottom left) and Cray XT6 (bottom right) for the 465K system. Nodes were fully populated in all benchmarks.

macs (3) and also the Desmond MD program (28) that energy conversion is at least as good as in double precision MD programs and therefore single precision programs yield results as accurately. The benchmarks clearly reveal that single precision can be up to 50% times faster. However, it must also be mentioned that double precision is required for e.g. normal mode analysis or very accurate energies.

Gromacs 4.5.3 achieves peak performance at 2048 cores. The number of PME nodes for the data shown in Figure 3 has been hand-optimised and we generally find that best performance can be obtained when about 25–30% of the cores are dedicated to computing long-range electrostatics with PME. At higher core counts, however, we found that for the floating point version it may be necessary to dedicate a larger number of cores to PME computations, e.g. performance was best when 352 cores were reserved for PME with 1024 cores total. The suggestion in the Gromacs output as to how to increase or decrease this number seems to become increasingly unreliable with increasing core numbers. Varying the number of PME tasks can have a dramatic effect on performance.

Benchmarking the new Cray XT6 (Figure 3, bottom right) using all cores per node were run for LAMMPS (May 29, 2010), NAMD 2.7b2 (only this version was available in the binary format specific for this platform) and GROMACS

4.5.3 (in double precision for fair comparison). The general picture is similar to the other platforms: LAMMPS scales slightly worse and performance peaks at 576 cores while NAMD peaks at 1536 cores. Gromacs attains peak performance earliest at 480 cores but is faster than NAMD. All numbers are multiples of 24 due to each Cray XT6 having 24 cores per node in contrast to the two other platforms where base 2 multiples were used.

3.4 Benchmarking the 3M system

Figure 4 summarises benchmarking results for the larger 3M system on the BlueGene/P, Cray XT4 and Cray XT6. The MD programs used were LAMMPS (May 29, 2010) and NAMD in versions 2.7b2 and 2.7b3 as available for the specific hardware. All benchmarks have been run with fully populated nodes except for NAMD on BlueGene/P because of the limited memory per node on this platform (25).

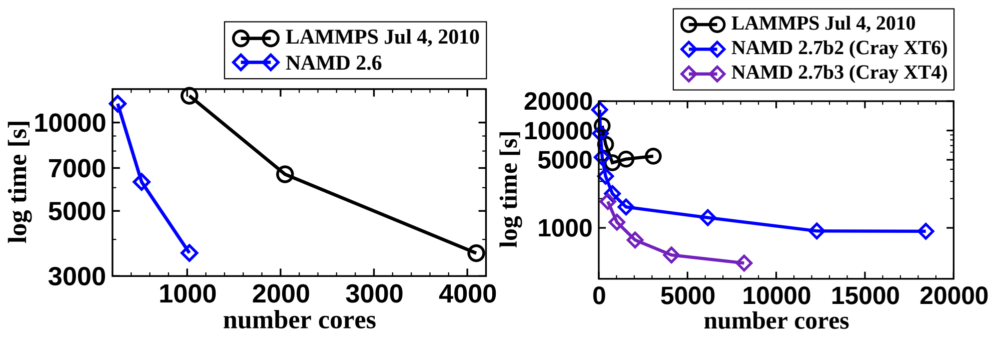

Fig. 4. Comparative benchmarks on the BlueGene/P (left) as well as the Cray XT4 and Cray XT6 (right) for the 3M system. Nodes were fully populated in all benchmarks except with NAMD on the BlueGene/P.

Not too much can be said for the BlueGene/P (Figure 4, left) as it was not possible to obtain more data points. NAMD could only be run for 1024 cores maximum and low cores in general would need too much computation time not available through the batch system. Generally, both programs scale to the maximum core count accessible. We would expect that positive scaling would increase beyond 4096 cores.

On the Cray XT6 NAMD scales up to about 18 000 cores although it is evident from Figure 4 (right) that performance essentially does not increase between 12 000 and 18 000 cores. It also appears that there is not much benefit in increasing the number of cores beyond 1536 cores when nodes are fully populated. LAMMPS' peak performance is at only 768 cores and this MD code also scales less well than NAMD. Scaling and also absolute performance is best with NAMD on the CRAY XT4 which may depend on the core number used per node.

To shed some light on the impact of multiple cores used per logical node we ran comparative simulations with non-fully populated nodes and the 465K system and a constant number of cores. On the BlueGene/P we find that LAMMPS performance increases by only about 1% when decreasing the cores/node from 4 to 2. When 1 core per node is used DL_POLY gains about 15–20% while NAMD 2.6 gains only 1–6% on the same platform. The latter percentage increases with increasing number of cores. On the Cray XT4 the percentage for LAMMPS and 4 vs. 2 cores/node is similar to the BlueGene/P and about 4% for 4 vs. 1 core/node.

The Cray XT6 can make use of even more cores per node (24) than the Cray XT4 and the BlueGene/P (4). What happens if we vary this number is shown in Figure 5 for both LAMMPS (May 29, 2010) and NAMD 2.7b2. LAMMPS' performance peaks at 576 cores with all nodes fully populated, i.e. 24 cores/node (data as in Figure 3 on the right). If we only allow 12 or 6 cores per node absolute performance and also scaling, up to 576 cores at least, improves strongly. It is important, however, to note that the total number of nodes doubles and quadruples respectively, and computation time on both phases of HECToR is charged per node! In the case of NAMD, performance reaches it peak at 1536 cores independent of the number of cores per node chosen. Scaling improves less well at lower number of cores per node but absolute performance certainly benefits considerably when hexacores are sparsely populated.

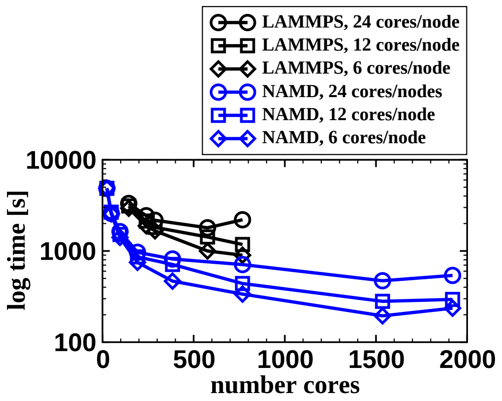

Fig. 5. LAMMPS (May 29, 2010) and NAMD 2.7b2 benchmarks on the CRAY XT6 for various populations schemes among the cores of a node.

Despite the increase in absolute performance, this practice of decreasing the number of cores per node is, from the economical view point at least, obviously not reasonable. However, performance can still be increased at a moderate cost or, in some cases, even reduced cost by understanding the hardware details of a node on the Cray XT6. Each node comprises four hexacores where each hexacore has uniform memory access to its own memory bank. Access to the memory banks of the other hexacores, in contrast, is non-uniform, i.e. may lead to increased latency and reduced bandwidth. This may be offset, however, by the increased number of memory accesses to the *same* local memory bank in case of a higher number of running processes per hexacore.

It is possible to control how many cores per hexacore will be populated and that may lead in turn to improved performance. For instance, within each node only 20 (or 16) cores could be populated in total but distribution among the hexacores may be required to run only 5 (or 4) processes per hexacore, that is the processes are evenly distributed among the 4 hexacores in contrast to filling up each hexacore to the maximum before attempting to fill up the next as is the default.

In Figure 6 we present benchmarks of the 465K system on the Cray XT6 with varying numbers of cores per node. All curves are plotted as functions of *nodes* to emphasise on the actual cost for each run. The thick black line was chosen as the reference and is the data for the fully populated runs. Benchmark data plotted to the bottom and the left of the reference represent thus runtimes that are faster *and* cheaper than the reference.

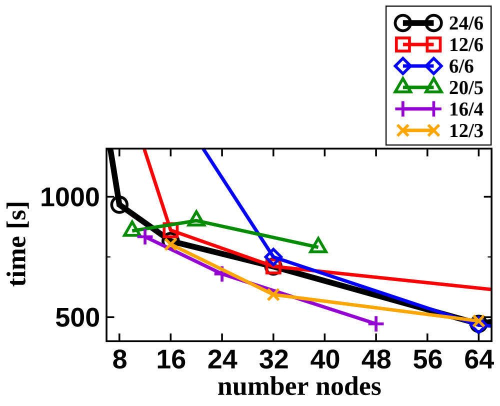

Fig. 6. Benchmarks with NAMD 2.7b2 on the CRAY XT6 and varying cores per hexacore. The 465K system was chosen as test system. Note that these graphs are plotted as a function of *nodes* rather than cores as in the other figures. The thick black line represents the runs with fully populated cores and is taken as the performance reference.

The choice of a 20/5 population (20 cores total, 5 per hexacore) is only beneficial for 10 nodes. 12/6 and 6/6 populations do not improve on the reference runs. 16/4 and 12/3 runs, on the other hand, do profit from the more sparsely distribution of processes among each hexacore. However, this may be only true for the node numbers presented in Figure 6. Other combinations of total cores/cores per hexacore and other node numbers may yield different performance results.

Since December 17, 2010 the upgraded Cray XT6, now XE6, is available to all users of the system although a two weeks acceptance test is due for mid January, 2011. Here we present some early benchmarking results with Gromacs 4.5.3 and NAMD 2.7 to assess the impact of the new Gemini communication interconnect on parallel performance. Gromacs has been recompiled by us as this was necessary to run on the new hardware. NAMD was compiled by the support team.

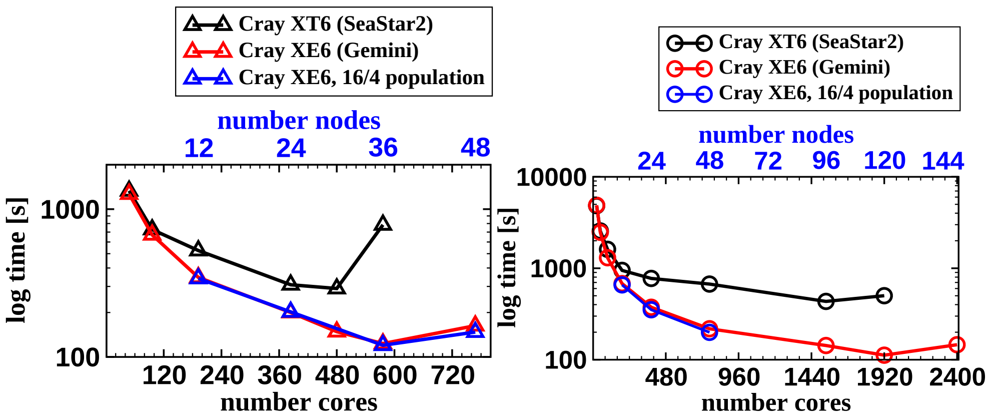

Fig. 7. Benchmarks with Gromacs 4.5.3 (left) and NAMD 2.7 (right) of the 465K system on the CRAYs XT6 and XE6. Runs with 16/4 population are also included.

In Figure 7 we plot the results for both the old SeaStar2 interconnect (Cray XT6) and the new Gemini interconnect (Cray XE6). It is obvious that the performance on the new hardware is improved over the older one. The 16/4 population run on the other hand demonstrates that the potential for further decreasing runtimes through under-population of hexacores seems limited (cf. Figure 6) although different populations and also assignments of numbers of PME nodes with Gromacs may alter the results. The number of PME nodes were the same as in the fully populated runs. NAMD profits from the hardware upgrade more than Gromacs.

3.7 Benchmarking the GPU cluster

Benchmark data for the GPU cluster are summarised on Table 3. Results are shown for the 465k system, varying the number of GPUs used, and the distribution of GPUs over nodes of the cluster. NAMD scales well with the number of GPUs used, with the distribution over nodes only having a minor effect, at least for this small test cluster. However, the real power comes from the parallelisation within a single GPU. For comparison, the same job running

Table 3
NAMD 2.7b3 running on the Daresbury GPU cluster "cseht".

No. nodes	GPUs/node	Time/s	ns/day
1	2	8136	0.223
1	4	4431	0.393
2	1	8217	0.230
2	2	4215	0.417
2	4	2581	0.675
4	1	4042	0.443
4	2	2354	0.758
4	4	1749	1.011

on 4 CPU cores, and making no use of the connected GPUs, took 21579s i.e. roughly 5 times slower than using 4 GPUs.

4 Summary

We have carried out molecular dynamics benchmarking simulations to assess the performance of various popular MD software programs on the HPC platforms BlueGene/P, Cray XT4 and Cray XT6/XE6. The MD packages considered were DL_POLY 4, Gromacs 4.0 and 4.5, LAMMPS in the most recent version available (daily development snapshots can be downloaded from their web page), and NAMD 2.6 as well as final and beta versions of 2.7 as available. We have found that the chosen MD programs display quite different performance characteristics. We will summarise a few general observations in this section and give advice on how to possibly improve performance.

DL_POLY scaled less well than LAMMPS and NAMD on the BlueGene/P although it is faster than LAMMPS on lower core counts. NAMD generally scaled well and performed faster but cannot beat the newest version of Gromacs. Gromacs achieves peak performance at lower core counts on the Cray XT6/XE6. Similar results were found recently in part I of this report series (1). On the Cray XT4 scaling is similar to NAMD but Gromacs runtimes are shorter.

In table 4 we summarise the peak performance found for all benchmarked HPC platforms and MD programs. It is expressed in terms of maximum achievable nanoseconds per day for convenience. Scalability to high core counts, however, is less important than high absolute performance achieved at low core counts

Table 4
Peak performances in ns/day achieved with the MD programs in this report on HPC platforms. All cores fully populated.

	BlueGene/P	Cray XT4	Cray XT6	Cray XE6 [a]
	465K system			
DL_POLY 4.0	1.0			
Gromacs 4.5 (float)	17.7	26.7		
Gromacs 4.5 (double)		23.1	5.9 [b]	14.0
LAMMPS	2.7 [c]	4.0	1.0	
NAMD 2.7	5.4 [d]	12.9	3.7	15.4
	3M system [e]			
LAMMPS	0.5		0.3	
NAMD 2.7	0.5 [f]	4.0	1.8	

[a] Like Cray XT6 but with Gemini communication interconnect.
[b] The number of PME nodes was not optimised for higher core numbers.
[c] Optimised routines.
[d] Non-MPI version.
[e] Limited number of cores and memory on BlueGene/P.
[f] Version 2.6 with MPI.

as Gromacs demonstrates. This code has been optimised for fast execution on individual CPUs and reaches minimal runtimes very early on. A performance of around 1 ns/day for the 465k system can be achieved with NAMD on a cluster of 16 GPUs (See Table 3), and this may be a viable alternative to the traditional HPC systems.

It is also important to point out that the region around the performance peak is rather flat which means that larger changes in the number of cores will only lead to small changes in runtimes. An extreme example is the NAMD benchmark of the 3M system on the Cray XT6 (see Figure 3, bottom right). Arguably, the peak performance was found at a node count of 768 (18432 cores) but essentially the same runtimes could be achieved with 512 nodes (12288) which means at two thirds of the cost (computational resources are paid per node on HECToR). However, if we use only 64 nodes runtimes would only increase by less than half but resulting in only a sixth of the cost of a 768 node run. From the performance vs. cost perspective it may therefore be beneficial to run simulations at node counts much less (typically 50%) than what would be needed for peak performance.

DL_POLY may, overall, perform less well than the other codes but it must be kept in mind that the code supports very general potential functions and is therefore not optimised for biomolecular simulations. LAMMPS has undergone changes in the programming languages used. Hence the code had probably less time to mature and there is still potential for optimisation as the separate long-range routines (see Figure 2) demonstrate. We expect future work to lead to improved performance for all major MD programs and more sophisticated parallelisation strategies will be part of the game.

Both LAMMPS and NAMD scaled reasonably well with the larger 3 million atom system on the BlueGene/P but the number of cores is limited to 4096. On HECToR phase 2b (CRAY XT6) we observed that LAMMPS' performance collapsed already at rather low core counts and the absolute performance is considerably smaller than the one of NAMD. To a lesser extent this is also true for the smaller system (ca. 465 000 atoms).

We have also found that the multi-core architecture of the CRAY XT6 poses particular problems for all MD programs leading to a lower performance than on the Cray XT4 (cf. Figure 4, right). Part of the issue is that each node had only one SeaStar2 communication interconnect though the installation of the new Gemini interconnect seems to alleviate the impact on performance. To reduce runtimes and at the same time keep costs within reasonable limits the user can try to distribute the load within a node by hand (via the -N and -S flags of `aprun`). We have found that an evenly distributed population of cores can lead to improved performance and the cost can be lower compared to fully populated runs (see Figure 6). As of yet we cannot make a general statement what population would be most beneficial. However, the performance of MD codes on the CRAY XE6 is currently not very impressive for Gromacs as the code runs still only at about 60% the speed of the CRAY XT4 when cores are fully populated. NAMD on the other hand profits considerably from the new interconnect which can outperform the Cray XT4.

So what can a user do to use MD software on a particular HPC platform as efficiently (both in time and money) as possible? (Re)compilation of the software with corresponding optimisation flags to the compiler may be possible but we can generally assume that the developers take care of that and such a procedure may only be useful on platforms with which the developers have less experience or when no official port exists. However, too aggressive optimisation may break the code (or even show up bugs!) and careful evaluations of the results produced must be undertaken. It is even possible that performance actually decreases. Generally, such a strategy is probable only helpful in a minority of cases.

More obvious efficiency gains may be achieved through careful system setup and simulation parameters. The number of water molecules, for instance, can be reduced but a sane minimum amount should be retained. Programs like

Gromacs and also Amber allow various box shapes one of which may be closer to the shape of the simulated system and thus may allow the reduction of the number of solvent. The most time consuming algorithm in an MD program is, however, the evaluation of the long range interactions and the variation of simulation parameters affecting this algorithm may lead to faster run times.

To alleviate the computational cost of long-range interaction evaluation NAMD allows the use of so-called multiple time steps algorithm. Practically, this means that long-range potentials and forces are evaluated less frequently than short-range interactions because the former are known to vary slower. In a twin-range approach this would typically mean to evaluate the long-range interactions only every second step. Cutoffs should also be kept relatively short with PME as these parameters determine patch sizes and hence the work load to be distributed.

The developers of Gromacs have identified the FFT (fast Fourier transformation) algorithm to be the current bottle neck in their MD code (see discussion in (3)). FFT algorithms need the charges to be distributed onto grids and therefore the grid size affects the performance. Reducing that size but at the same time increasing the PME order may lead to faster run-times at comparable accuracy. The number of PME nodes must be adjusted by hand. We found that reserving about 25–30% for this tasks leads to best performance. The tool `g_tune_pme` in the newest Gromacs distribution may be useful to adjust PME parameters automatically. Finally, general improvements in the code will be necessary to scale to the hundred thousands (or even more) of cores of HPC platforms in the future.

Acknowledgement. We acknowledge support by HECToR, by the CSED of STFC for access to its IBM Blue Gene/P system and by the DisCo group for access to their test GPU cluster. We are especially grateful to the user support teams of these systems. We also thank Dr. Chin W. Yong for carrying out the DL_POLY benchmarks.

References

[1] H. H. Loeffler, M. D. Winn, Large biomolecular simulation on HPC Platforms I. Experiences with AMBER, Gromacs and NAMD, Tech. Rep. DL-TR-2009-002, STFC Daresbury Laboratory, Warrington WA4 4AD, UK (2009).
URL `http://epubs.stfc.ac.uk/work-details?w=50963`

[2] D. A. Case, T. E. Cheatham III, T. Darden, H. Gohlke, R. Luo, K. M. Merz Jr., A. Onufriev, C. Simmerling, B. Wang, R. J. Woods, The Amber biomolecular simulation programs, J. Comput. Chem. 26 (2005) 1668.

[3] B. Hess, C. Kutzner, D. van der Spoel, E. Lindahl, Gromacs 4: Algorithms for highly efficient, load-balanced, and scalable molecular simulation, J. Chem. Theory Comput. 4 (2008) 435.

[4] J. C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. D. Skeel, L. Kalé, K. Schulten, Scalable molecular dynamics with NAMD, J. Comput. Chem. 26 (2005) 1781.

[5] S. Plimpton, Fast parallel algorithms for short-range molecular dynamics, J. Comput. Phys. 117 (1) (1995) 1 – 19.

[6] S. J. Plimpton, R. Pollock, M. Stevens, Particle-mesh ewald and rRESPA for parallel molecular dynamics simulations, in: Eighth SIAM Conference on Parallel Processing for Scientific Computing, Minneapolis, MN, 1997.

[7] W. Smith, I. T. Todorov, A short description of DL_POLY, Mol. Sim. 32 (12) (2006) 935–943.

[8] I. Todorov, W. Smith, The DL_POLY_4.0 User Manual, STFC Daresbury Laboratory, Daresbury, Warrington WA4 4AD, Cheshire, UK (October 2010).
URL `http://www.ccp5.ac.uk/DL_POLY/`

[9] J. A. Anderson, C. D. Lorenz, A. Travesset, General purpose molecular dynamics simulations fully implemented on graphics processing units, J. Comput. Phys. 227 (2008) 5342–5359.

[10] M. J. Harvey, G. Giupponi, G. De Fabritiis, Acemd: Accelerating biomolecular dynamics in the microsecond time scale, J. Chem. Theory Comput.

[11] J. E. Stone, J. C. Phillips, P. L. Freddolino, D. J. Hardy, L. G. Trabuco, K. Schulten, Accelerating molecular modeling applications with graphics processors, J. Comput. Chem. 28 (2007) 2618–2640.

[12] J. Kästner, H. H. Loeffler, S. K. Roberts, M. L. Martin-Fernandez, M. D. Winn, Ectodomain orientation, conformational plasticity and oligomerization of ErbB1 receptors investigated by molecular dynamics, J. Struct. Biol. 167 (2) (2009) 117 – 128.

[13] C. C. Gruber, J. Pleiss, Systematic benchmarking of large molecular dynamics simulations employing gromacs on massive multiprocessing facilities, J. Comput. Chem. (2010) (early view).
URL `http://dx.doi.org/10.1002/jcc.21645`

[14] A. D. MacKerell, D. Bashford, Bellott, R. L. Dunbrack, J. D. Evanseck,

M. J. Field, S. Fischer, J. Gao, H. Guo, S. Ha, D. Joseph-McCarthy, L. Kuchnir, K. Kuczera, F. T. K. Lau, C. Mattos, S. Michnick, T. Ngo, D. T. Nguyen, B. Prodhom, W. E. Reiher, B. Roux, M. Schlenkrich, J. C. Smith, R. Stote, J. Straub, M. Watanabe, J. Wiorkiewicz-Kuczera, D. Yin, M. Karplus, All-atom empirical potential for molecular modeling and dynamics studies of proteins, J. Phys. Chem. B 102 (18) (1998) 3586–3616.
URL `http://pubs.acs.org/doi/abs/10.1021/jp973084f`

[15] A. D. Mackerell, M. Feig, C. L. Brooks, Extending the treatment of backbone energetics in protein force fields: Limitations of gas-phase quantum mechanics in reproducing protein conformational distributions in molecular dynamics simulations, J. Comput. Chem. 25 (11) (2004) 1400–1415.
URL `http://dx.doi.org/10.1002/jcc.20065`

[16] S. E. Feller, A. D. MacKerell, An improved empirical potential energy function for molecular simulations of phospholipids, J. Phys. Chem. B 104 (31) (2000) 7510–7515.
URL `http://pubs.acs.org/doi/abs/10.1021/jp0007843`

[17] S. E. Feller, K. Gawrisch, A. D. MacKerell, Polyunsaturated fatty acids in lipid bilayers: Intrinsic and environmental contributions to their unique physical properties, J. Am. Chem. Soc. 124 (2) (2002) 318–326, pMID: 11782184.
URL `http://pubs.acs.org/doi/abs/10.1021/ja0118340`

[18] D. Beglov, B. Roux, Finite representation of an infinite bulk system: Solvent boundary potential for computer simulations, J. Chem. Phys. 100 (12) (1994) 9050–9063.
URL `http://link.aip.org/link/JCPSA6/v100/i12/p9050/s1`

[19] W. L. Jorgensen, J. Chandrasekhar, J. D. Madura, R. W. Impey, M. L. Klein, Comparison of simple potential functions for simulating liquid water, J. Chem. Phys. 79 (2) (1983) 926–935.
URL `http://link.aip.org/link/?JCP/79/926/1`

[20] P. Bjelkmar, P. Larsson, M. A. Cuendet, B. Hess, E. Lindahl, Implementation of the charmm force field in gromacs: Analysis of protein stability effects from correction maps, virtual interaction sites, and water models, J. Chem. Theory Comput. 6 (2) (2010) 459–466.
URL `http://pubs.acs.org/doi/abs/10.1021/ct900549r`

[21] O. Berger, O. Edholm, F. Jähnig, Molecular dynamics simulations of a fluid bilayer of dipalmitoylphosphatidylcholine at full hydration, constant pressure, and constant temperature, Biophys. J. 72 (1997) 2002.

[22] H. J. C. Berendsen, J. P. M. Postma, W. F. van Gunsteren, J. Hermans, in: B. Pullman (Ed.), Intermolecular Forces, Reidel, Dordrecht, 1981, p. 331.

[23] U. Essmann, L. Perera, M. L. Berkowitz, T. Darden, H. Lee, L. G. Pedersen, A smooth particle mesh ewald method, J. Chem. Phys. 103 (19) (1995) 8577–8593.
URL `http://link.aip.org/link/?JCP/103/8577/1`

[24] R. W. Hockney, J. W. Eastwood, Computer Simulation Using Particles, Taylor & Francis, 1989.

[25] K. Y. Sanbonmatsu, C.-S. Tung, High performance computing in biology: Multimillion atom simulations of nanoscale systems, J. Struct. Biol. 157.

[26] W. Humphrey, A. Dalke, K. Schulten, VMD – Visual Molecular Dynamics, J. Mol. Graphics 14 (1996) 33–38.

[27] C. W. Yong, The DL_FIELD Software Package, STFC Daresbury Laboratory, Daresbury, Warrington WA4 4AD, Cheshire, UK (November 2010). URL `http://www.cse.scitech.ac.uk/ccg/software/DL_FIELD/`

[28] K. J. Bowers, E. Chow, H. Xu, R. O. Dror, M. P. Eastwood, B. A. Gregersen, J. L. Klepeis, I. Kolossvary, M. A. Moraes, F. D. Sacerdoti, J. K. Salmon, Y. Shan, D. E. Shaw, Scalable algorithms for molecular dynamics simulations on commodity clusters, SC Conference 0 (2006) 43.

5 Appendix

Here we provide a summary of the input files used for the benchmark studies.

(1) NAMD config file

```
set p          1.0
set T          300.0
set num_steps  10000

set new       cores$env(NAMD_CORES)
set previous  relres

# CHARMM force field
structure          H1H1.psf
paratypecharmm     on
parameters         par_all27_prot_lipid.prm
coordinates        H1H1.pdb
exclude            scaled1-4
1-4scaling         1.0

# restart options
binvelocities        $previous.vel
bincoordinates       $previous.coor
ExtendedSystem       $previous.xsc
firsttimestep        0

# temperature coupling
langevin              on
langevinHydrogen      on
langevinTemp          $T
langevinDamping       1

# pressure coupling
useGroupPressure      yes

LangevinPiston        on
LangevinPistonTarget  $p
LangevinPistonPeriod  200
LangevinPistonDecay   100
LangevinPistonTemp    $T

# output
outputname          $new
```

```
outputEnergies      1000
outputPressure      1000
outputTiming        10000
binaryoutput        yes

# write restart
restartname         $new
restartfreq         1000
binaryrestart       yes

# trajectory file
DCDfile             $new.dcd
DCDUnitCell         yes
DCDfreq             5000

# eXtended System Trajectory
XSTfile             $new.xst
XSTfreq             1000

# wrap coordinates
wrapAll             on
wrapNearest         on

# multipe time step settings
nonbondedFreq       1
fullElectFrequency 1

# no rigid bonds
useSettle           on
rigidBonds          all
rigidIterations     100

# cutoffs and non-bonded pair lists
switchdist          10
cutoff              12
switching           on
pairlistdist        14
pairlistsPerCycle   1
stepspercycle       20

# particle mesh ewald
PME                 on
PMEGridSpacing      1.0
```

```
timestep 2.0 # in fs

run $num_steps
```

(2) Gromacs .mdp file

```
title                    = Her1-Her1 in POPC bilayer
cpp                      = /usr/bin/cpp

integrator               = md
tinit                    = 0.0
dt                       = 0.002
nsteps                   = 10000
nstcomm                  = 1000

nstxout                  = 5000
nstvout                  = 0
nstfout                  = 0
nstlog                   = 1000
nstenergy                = 1000
nstxtcout                = 0
xtc_precision            = 1000

nstlist                  = 10
ns_type                  = grid
pbc                      = xyz
rlist                    = 1.2

coulombtype              = PME
rcoulomb                 = 1.2

fourierspacing           = 0.12
fourier_nx               = 0
fourier_ny               = 0
fourier_nz               = 0
pme_order                = 4
ewald_rtol               = 1.0E-5
optimize_fft             = yes

vdwtype                  = Shift
rvdw_switch              = 1.0
rvdw                     = 1.1
DispCorr                 = no

tcoupl                   = Berendsen
```

```
tc_grps                    = System
tau_t                      = 1.0
ref_t                      = 300.0

pcoupl                     = Berendsen
pcoupltype                 = isotropic
tau_p                      = 2.0
compressibility            = 4.5e-5
ref_p                      = 1.0

gen_vel                    = no

constraints                = none
constraint_algorithm       = Lincs
unconstrained_start        = no
lincs_order                = 4
lincs_warnangle            = 30
```

(3) LAMMPS input file

```
variable          T equal 300.0
variable          p equal    1.0
variable          Tdamp equal   100.0
variable          pdamp equal 1000.0

variable          old string "prep3"
variable          new string "@LAMMPS_JOBNAME@"

units             real
neigh_modify      delay 0 every 1 check yes

atom_style        full
bond_style        harmonic
angle_style       charmm
dihedral_style    charmm
improper_style    harmonic

pair_style        lj/charmm/coul/long 10 12
pair_modify       mix arithmetic
kspace_style      pppm 1e-4

read_restart      ${old}.rst.*

special_bonds     charmm
fix               1 all shake 1e-6 500 0 m 1.0 b 49 a 111
fix             2 all npt temp $T $T ${Tdamp} iso $p $p ${pdamp}

thermo          1000
thermo_style    multi
thermo_modify   flush yes
timestep        2.0

restart         1000 ${new}.rst
dump            1 all dcd 5000 ${new}.dcd
dump_modify     1 unwrap yes

run             10000
```