PERFORMANCE STUDY OF MOLECULAR DYNAMICS APPLICATION GROMACS

Zongjie Cui and Zhiling Lan
Department of Computer Science
Illinois Institute of Technology
10 W 31st Street
Chicago, Illinois USA 60616
{cuizong, lan}@iit.edu

ABSTRACT
One of the most exciting and difficult challenges in biology area is to understand the interactions of complex biological systems. Computational simulations have become increasingly important in enabling progress in biological research. GROMACS is such a software package that provides a variety of tools to enable biological simulations. In this paper, we investigate and analyze the performance of GROMACS in two widely used cluster environments: a Linux cluster and a Unix cluster. The performance is studied from three aspects: overall performance, communication characteristics, and load balancing characteristics. Our analysis shows that GROMACS suffers from three major performance issues: (1) it cannot scale well beyond 16 processors; (2) the underlying communication scheme of GROMACS is not efficient; and (3) it does not include any dynamic load balancing scheme for efficient execution on parallel and distributed systems. Besides analyzing the application and locating the bottlenecks, we also provide several candidate solutions to improve the performance of GROMACS on large-scale cluster systems.

KEY WORDS
Performance evaluation, High performance computing, Cluster computing, Message passing, Molecular dynamics

1. Introduction

Computational simulations have become increasingly important in enabling progress in biological research. In recognition of the critical role computational simulations play in the research of molecular dynamics, a group of scientists developed GROMACS, a software package that provides a variety of tools to enable biological simulations [1]. GROMACS is primarily designed for biochemical molecular dynamics that have a lot of complicated bonded interactions, now it is extended with interfaces to both quantum chemistry and bioinformatics. GROMACS is the only molecular dynamics code that has been released under GNU General Public License. It is one of the most popular molecular dynamics codes used by scientists and is about to become the new molecular dynamics driver for Folding@Home [6]. Furthermore, GROMACS is a computing-intensive application in that a typical GROMACS simulation requires a large amount of computing power. For example, simulations and statistical mechanical analysis of ion channels requires more than several days running on a 128-node parallel machine.

Besides GROMACS, there are several other molecular dynamics packages available. Among them, NAMD [22, 23], a software package based on Charm++[24], is one of the most successful packages. According to the comparison work done by Peter Spijker [21], NAMD is more focused on biological simulations such as proteins or lipid bilayers, while GROMACS is more aimed for computational chemistry on a whole and is very useful for coarse grain simulations. He also points that GROMACS is faster in terms of calculation. As GROMCS is under GNU General Public License, we believe that a detailed performance evaluation and optimization of GROMACS can benefit a broader community.

The developers of GROMACS have made a great effort to improve the performance of GROMACS, such as various algorithm optimizations, utilizing specific hardware instruction sets, and using Message Passing Interface (MPI) for better performance on parallel systems. These optimizations make GROMACS the fastest scalar molecular dynamics code. However, our analysis indicates that GROMACS does not show a satisfying scaling on large-scale systems. In other words, with the increasing number of computing processors, the performance does not improve correspondently. It is critical to identify the performance bottlenecks and further to resolve them so as to take advantages of the parallelism provided by modern high performance computers.

Nevertheless, there is little work on performance analysis of GROMACS on parallel systems. Some performance data are shown in the GROMACS website [1]; however, they do not provide much detail in terms of identifying performance bottlenecks. Further, few work has been done in terms of evaluating GROMACS on clusters which are the emerging high performance computing platform. In this paper, we perform a detailed

performance analysis of GROMACS on widely used clusters from several aspects including overall performance evaluation, communication characteristics, and load balancing characteristics. Both benchmarking and real-life datasets are evaluated on different cluster platforms. Our analysis shows that GROMACS suffers from three major performance issues: (1) it cannot scale well beyond 16 processors; (2) the underlying communication scheme of GROMACS is not efficient; and (3) it does not include any dynamic load balancing scheme for efficient execution on parallel and distributed systems.

The rest of the paper is organized as follows. Section 2 gives an overview of GROMACS. Section 3 describes the detailed configuration of our experiments including our approaches, the experimental environments and the datasets. Section 4 presents our performance data. In Section 5, we propose solutions to solve these performance problems associated with GROMACS. Lastly, we summarize the paper and describe our future work in Section 6.

2. Molecular Dynamics Application GROMACS

Molecular Dynamics (MD) is a numerical simulation technique where the time evolution of a set of interacting atoms is followed by integrating their equations of motion [15]. MD is widely used in the area of computational chemistry for discovering complex chemical systems in terms of realistic atomic models. The backbone of MD includes: 1) Determine the initial parameters like starting positions, velocities, and masses of the particles. 2) Distribute the workload (i.e. the particles) to every processor if MD is run in a parallel computing environment. 3) Determine the maximum number of steps or ending condition. 4) Calculate the forces between particles. 5) Let the particles move in the experienced force field for the time defined before. 6) Repeat 4-5, called the MD loop, until meeting the ending conditions. 7) Do some finalizing jobs such as generating the output files. To make it simpler and clearer, we divide the execution of GROMACS into three parts: before the MD loop, within the MD loop, and after the MD loop.

In order to parallelize the MD algorithm, there are a number of techniques. Message Passing and Data Parallelism are two commonly used methods. For many concerns including portability, GROMACS chooses the former one: Message Passing. In GROMACS, all communications among computing nodes are achieved through Message Passing Interface (MPI) library. Moreover, in order to achieve high performance not only on those supercomputers with luxurious inter-processor connections but also on those commodities workstation-based or PC-based clusters, GROMACS uses a simple ring topology. Each processor in the ring gets a slab of the box in the X-dimension. During each step of the MD loop, most portions of communication and information exchanging happen between neighbours (left and right processors) except for a small number of global communications in the PME (Particle-Mesh Ewald, a method proposed by Tom Darden [18, 19] to improve the performance of the reciprocal sum) module.

3. Experiments

3.1 Experimental Configuration

In order to get the accurate performance of GROMACS, we manually instrument the code. Our instrumentation is designed carefully with the goal not to cause much overhead. For instance, it does not require any extra communication through networks and only one I/O operation is required at the end of simulation to generate the output of performance data. The performance data being collected includes: number of MPI calls, execution time of each MPI call, message size of each MPI call, aggregated time of all MPI calls, aggregated message size of all MPI calls, computation time, total execution time, and workload (number of atoms) per processor. As mentioned earlier, a typical MD simulation consists of three major parts. The second part, the MD loop, is the most time-consuming part while the other two do some necessary jobs that have little potential for performance tuning. Therefore, we separate the data of these stages so that we can focus on the MD loop part. At runtime, each process collects its own performance data and generates a unique log file. To get the summary of the entire run, we develop separate tools to generate the statistic data including the minimum, the maximum, the mean value, and the standard deviation for each item.

Two cluster systems are chosen for this work: one is an IA-64 Linux cluster Titan at the National Center for Supercomputing Applications (NCSA) [5] and the other is a SUN cluster Sunwulf at the Scalable Computing Software Laboratory of Illinois Institute of Technology. We choose these systems due to several reasons. First of all, they are both typical cluster systems widely used in the area of scientific computing. Secondly, they stand for two different types of clusters: one for Linux PC clusters and the other for Unix Network of Stations (NOW). Lastly, they have different configurations of hardware, compiler, operating systems, MPI and other software environments. This provides us with a pretty fair performance evaluation of GROMACS.

3.2 Experimental Datasets

The performance of GROMACS depends on not only the intrinsic characteristics of the application, but also the input dataset files. In order to get rid of the volatile impacts of any arbitrary input dataset, we analyze one real-life dataset and two benchmarking datasets with varying number of computing nodes and different input parameters. In general, biologists are particularly interested in the comparison of performance with/without

PME. Therefore, we choose these datasets such that some of experiments use PME while others do not.

The first dataset we used is a real-world dataset (denoted as DCS) provided by the biological research group [25] at Illinois Institute of Technology. This dataset is used for studying the properties of biological membranes through molecular dynamics. In such a simulation, three kinds of lipid (totally 284633 atoms) – 1424 DPOC particles (54 atoms/lipid), 122 cholesterol particles (51 atoms/lipid), and 266 sphingomyelin particles (52 atoms/lipid) – interact with 62561 water molecules (3 atoms/molecule). Within the each iteration of the MD loop, due to the interaction, some particles might move from its original computing box to its neighbour, resulting in a workload transfer among processors.

Besides the above real-world dataset, we also analyze two MD benchmarks provided by the developers of GROMACS. Lys/Cut simulates the larger lysozyme protein (totally 23207 atoms) by using cut-off method. DPPC simulates a phospholipid membrane consisting of 1024 dipalmitoylphosphatidylcholine (DPPC) lipids in a bilayer configuration with 23 water molecules per lipid (totally 121,856 atoms). Detailed information of both datasets can be found in the official website of GROMACS [1].

Totally, we have conducted six experiments: DCS (2000 iterations on Titan and 500 iterations on Sunwulf), DPPC (2000 iterations on Titan and 1000 iterations on Sunwulf), and Lys/Cut (2000 iterations on Titan and 1000 iterations on Sunwulf). Note that we use smaller number of iterations on Sunwulf due to the extremely long execution time on this platform. Further, due to some bugs of GROMACS version 3.1.4, we have problems to analyze certain datasets on some configurations. For instance, the simulation of Lys/Cut crashes when running on Titan with the configuration of 16, 24, 32, 48 and 64 processors.

4. Performance Data

GROMACS is analyzed from three aspects: overall performance and scalability, MPI communication, and dynamic load balancing characteristics.

4.1 Overall Performance and Scalability

In all of our experiments, we record the execution time of all the three stages: before, within and after the MD loop. Since the MD loop is the most time-consuming part, therefore we only show the performance data of the MD loop in the rest of the paper.

Figures 1-6 show the overall execution time of the MD loop for the six experiments. Here, the execution time is divided into computation part and communication part. Apparently, with the increasing number of computing nodes, the computation time decreases while the communication time increases. This makes sense because a larger number of computing nodes will accelerate the computation while it also introduces more synchronization and communication overhead. However, from these figures, the increasing speed of communication time is much faster than the decreasing speed of computation time, which results in an unsatisfied speedup, especially when the number of processors is more than 16, the efficiency is less than 50% in most cases. Obviously, this is a scalability problem. Our further investigation as shown in the following subsection indicates that it is caused by inefficient communication and load balance methods.

4.2 MPI Communication

One of the most important approaches to improve the scalability of GROMACS is to study its communication mechanisms. The underlying communication of GROMACS is built upon MPI mechanism. Therefore, how to efficiently utilize MPI in GROMACS is an important way to solve its scalability problem and further improve its overall performance. An effective way is to identify the hotspots of MPI communication used in GROMACS. There are totally 11 MPI function calls used in the GROMACS package: MPI_Isend, MPI_Send, MPI_Irecv, MPI_Recv, MPI_Wait, MPI_Test, MPI_Sendrecv, MPI_Allreduce, MPI_Abort, MPI_Bcast, and MPI_Reduce. All of them are wrapped in a network module, except that MPI_Bcast and MPI_Reduce are directly called in the PME [18, 19] module. Not all of them are called for every simulation. For example, MPI_Bcast and MPI_Reduce are used only when PME module is used.

Our instrument code records how many times each MPI function is called, and the results show MPI_Wait is the hotspot. This is caused by the large number of non-blocking MPI communications. When we further look at the code, we realize that non-blocking communication is not effectively utilized in GROMACS, which actually deteriorates the overall performance.

Figures 7-12 summarize the aggregated time of MPI calls. It is shown that MPI_Reduce and MPI_Bcast are the hotspots if PME is used, while MPI_Wait and MPI_Sendrecv are the hotspots without PME. MPI_Reduce is a global communication in which all the processors are involved for synchronization. The more computing nodes, the more possibility MPI_Reduce consumes more time. Therefore, in order to improve the scalability of GROMACS on parallel systems, how to reduce synchronization cost is obviously important.

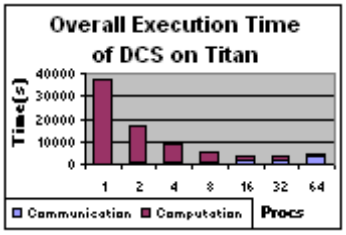

Figure 1

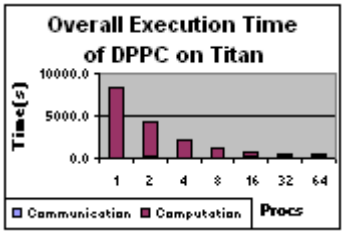

Figure 2

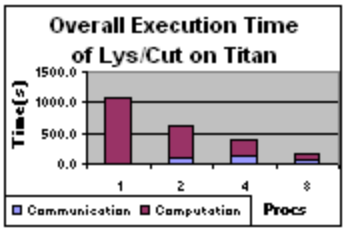

Figure 3

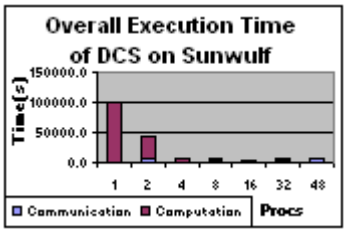

Figure 4

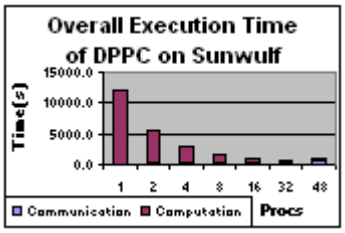

Figure 5

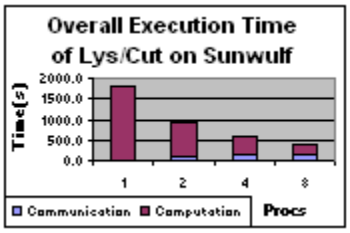

Figure 6

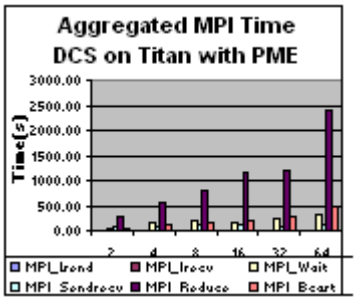

Figure 7

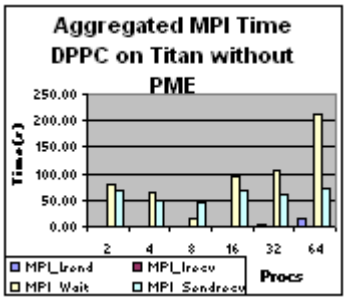

Figure 8

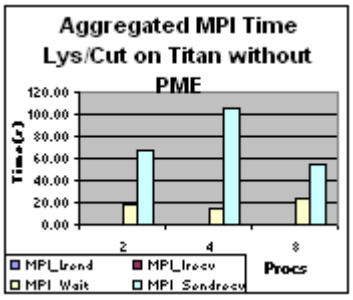

Figure 9

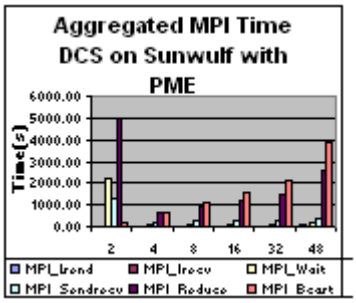

Figure 10

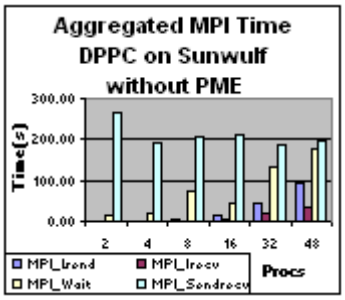

Figure 11

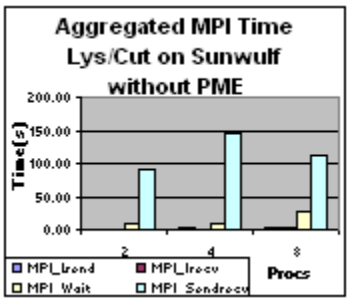

Figure 12

Message size is another important characteristic in MPI communication. Our experiments show that MPI_Sendrecv transfers the largest amount of data as compared to other MPI calls in most cases. This is caused by the nature of MD communication and is hard to improve. Nevertheless, redesigning the data structures for communication so as to reduce the message size is one of the possible improvements. MPI_Bcast and MPI_Reduce also deliver a large amount of messages when PME is used, while MPI_Isend and MPI_Irecv deliver a large amount of messages when PME is not used. Furthermore, when the number of nodes reaches 48 or 64, MPI_Isend and MPI_Irecv are the MPI calls that transfer the largest amount of message compared to all the others. In fact, both MPI_Isend and MPI_Irecv are used for point-to-point communication between neighbour nodes, which means arranging the underlying communication in a ring topology is not efficient for large-scale systems with more than 48 nodes. Hence, other topologies and communication methods should be considered for future performance improvement in GROMACS.

4.3 Load Balancing Characteristics

Load balancing is an important aspect in performance optimization of GROMACS running on parallel systems because it can reduce the difference in execution time between processors and eventually improve the efficiency of utilizing computing resources provided by parallel systems. The current version of GROMACS only performs static load balancing once at the beginning of computation. That is, the pre-processor grompp provides the options '-shuffle' and '-sort' to equally distribute workload among all the processors and sort the molecules according to their coordinates. GROMACS is an adaptive application in that its computational load

Mean and Standard Deviation of Computation Time (sec) of *Lys/Cut* on *Titan* and *Sunwulf*						
Proc Num	2 –*Titan*	4 – *Titan*	8 – *Titan*	2 – *Sunwulf*	4 – *Sunwulf*	8 – *Sunwulf*
Min	465.94	228.50	87.20	792.16	386.63	197.03
Max	585.40	367.11	148.80	898.11	528.92	318.17
Mean	525.68	268.73	96.68	845.14	429.93	222.80
St-Dev	59.72	57.49	19.78	52.97	58.33	36.87

Table 1

varies throughout the execution and causes uneven distribution of workload at run-time even though grompp can initially make workloads evenly distributed. In other words, the two options from grompp do not work well in some cases.

Figure 13 and Table 1 give such an example. Figure 13 shows the minimum and maximum computation time for Lys/Cut with varying number of processors (2, 4, and 8) on Titan and Sunwulf, while Table 1 summarizes the min, max, mean and standard deviation of computation time. Take the experiment of Lys/Cut with 8 processors on Titan as an example; the maximum computation time is 148.0(sec) while the minimum computation time is 87.20(sec). This clearly indicates that a serious load imbalance exist among the processors. The difference in terms of computation time among the processors is caused by the unbalanced workload. Therefore, dynamic load balancing (DLB) is an indispensable method to allow such an application to run efficiently on parallel systems.

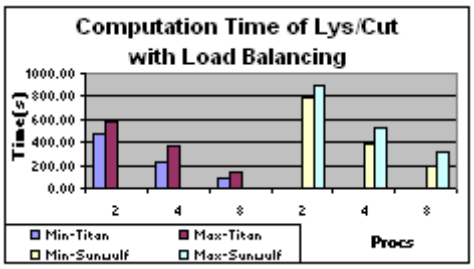

Figure 13

5. Analysis Results and Proposed Solutions

Based on the experimental results and our analysis above, we list the analysis results and propose several solutions for performance improvement of GROMACS on parallel systems as below.

1. Scalability Issue. Our experiments show that GROMACS cannot scale beyond 16 processors. This is mainly because of inefficient communication that will be addressed next. Further, in GROMACS, the available processors are organized as a ring. This is good for neighbouring communication. However, the modern cluster systems are organized in different topologies such as tree. Currently, we are working on how the underlying cluster organization will impact on the overall performance of GROMACS and other topologies (such as tree, 2D torus, and hypercube) are considered in our on-going research work with GROMACS.

2. Communication Inefficiency. We notice that there are several inefficient MPI communications in GROMACS. First of all, non-blocking MPI calls are not effectively utilized in GROMACS. Our experiments show that a large number of non-blocking calls are used in GROMACS. However, when looking into the source code of mdrun, we realize that each non-blocking MPI call is immediately followed by an MPI_Wait function. This is not an efficient way to utilize non-blocking communication provided by MPI. The solution is to overlap certain amount of computation with the non-blocking communication such that the computation and the MPI calls can run in parallel. Secondly, MPI_Reduce is the performance bottleneck in terms of execution time if the PME module is used. As it is a type of global communication, it probably cannot be easily replaced with other MPI calls due to the nature of the operation. Therefore an improved implementation of MPI_Reduce on cluster environments is one possible solution to remove such a bottleneck. Lastly, in the current implementation of GROMACS, only 11 MPI function calls (mostly point-to-point communication between direct neighbours) are utilized. In fact, MPI supports hundreds of functions each with various performance emphases. Different types of MPI calls including the use of collective communication and communicators are explored in our current work.

3. Lack of Dynamic Load Balancing. The current version of GROMACS provides a simple static load balancing mechanism for workload distribution among the processors on parallel systems. However, GROMACS is an adaptive application in that its computational load changes dynamically at runtime resulting in load imbalance. As shown in our experiments, severe load imbalance can occur among the processors. Therefore, an efficient dynamic load-balancing algorithm is required to keep the workload equally distributed among the processors. It will also solve the scalability problem associated with GROMACS.

6. Conclusions and Future Work

In this paper, we present a detailed performance study of GROMACS, a widely used molecular dynamics application, with both the real-life dataset and benchmark datasets on two widely used clusters systems. We analyze the performance characteristics of GROMACS from three aspects: overall performance evaluation, communication characteristics, and load balancing characteristics. Our study shows that GROMACS works well on cluster systems with a small number of processors, but has a serious scalability problem on a larger system with more

than 16 processors. Secondly, we notice that the communication of GROMACS is not efficient and several performance bottlenecks are identified. Furthermore, we notice that the simple static load balancing method in GROMACS cannot efficiently distribute the workload among the processors. Several solutions are proposed to address these performance issues.

Currently, we are working on investigating different solutions to improve the performance of GROMACS on parallel systems. Furthermore, we would like to extend GROMACS to the emerging Computational Grid [20] so as to tackle those problems that require vast computing power (e.g., beyond that available at any single site).

7. Acknowledgements

This work is supported by a grant from the National Computational Science Alliance with the NSF PACI Program and is supported in part by Illinois Institute of Technology ERIF grant.

References:

[1] GROMACS Official Web Site. http://www.gromacs.org.

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

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

[4] GROMACS User Manual. Version 3.1.1.

[5] Alliance. IA-64 and IA-32Linux Cluster at NCSA. World Wide Web, http://www.ncsa.uiuc.edu/.

[6] Folding@home. http://folding.stanford.edu/

[7] LAM/MPI Web Site. http://www.lam-mpi.org.

[8] Message Passing Interface Forum. http://www-unix.mcs.anl.gov/mpi/.

[9] K.J. Barker and N.P. Chrisochoides. An evaluation of a framework for the dynamic load balancing of highly adaptive and irregular parallel applications. In Proc. Of SC2003.

[10] T.W. Clark, R.V. Hanxleden, J.A. McCammon and L.R. Scott. Parallelizing molecular dynamics using spatial decomposition. In Scalable High-Performance Computing Conference, 1994.

[11] R. Hayashi and S.Horiguchi. Efficiency of dynamic load balancing based on permanent cells for parallel molecular dynamics simulation. In Parallel and Distributed Processing Symposium, 2000.

[12] D.F. Hegarty and M.T. Kec hadi. Topology preserving dynamic load balancing for parallel molecular simulations. In Proceedings of Supercomputing '97, 1997.

[13] R.K. Brunner, J.C. Phillips and L.V. Kale. Scalable molecular dynamics for large biomolecular systems. In Proceedings of the 2000 ACM/IEEE SC2000 Conference. ACM, 2000.

[14] D.M. Beazley and P.S. Lomdahl. Lightwright computational steering of very large scale molecular dynamics simulations. In Proceedings of the 1996 ACM/IEEE conference on Supercomputing (CDROM).

[15] F. Ercolessi. A molecular dynamics primer. http://www.fisica.uniud.it/~ercolessi/md/md/.

[16] Q. Ma, J.A. Izaguirre and R.D. Skeel. Nonlinear instability in multiple time stepping molecular dynamics. In Proceedings of the 2003 ACM symposium on Applied computing.

[17] A. Mink and C. Bailly. Parallel implementation of a molecular dynamics simulation program. In Proceedings of the 30th conference on Winter simulation.

[18] Darden T, York D, Pedersen L (1993) J Chem Phys 98:10089–10092.

[19] Essman U, Perera L, Berkowitz ML, Darden T, Lee H, Pedersen LG (1995) J Chem Phys 103:8577–8592.

[20] I. Foster and C. Kesselman. The Grid: Blueprint for a New Computing Infrastructure", Morgan Kaufmann Publishers. 1999.

[21] Peter Spijker. NAMD and GROMACS: The Quest To Go Parallel.

[22] James C. Phillips, Gengbin Zheng, Sameer Kumar, Laxmikant V. Kalé. NAMD: biomolecular simulation on thousands of processors. In Proceedings of the 2002 ACM/IEEE conference on Supercomputing.

[23] NAMD Web Site. http://www.ks.uiuc.edu/Research/namd/.

[24] L. V. Kale and Sanjeev Krishnan. Charm++: Parallel Programming with Message-Driven Objects. Book Chapter in "Parallel Programming using C++", by Gregory V. Wilson and Paul Lu. MIT Press, 1996. pp 175-213.

[25] H.L. Scott, E.Jackobsson, and S.Subramaniam, “Simulation of Lipid Membranes with Atomic Resolution”, Journal of Computational Physics, 12:328--334.