Natural Load Indices (NLI) in NAMD2 load balancing algorithms

Stefan Muszala
Gita Alaghband
James J. Hack
Daniel Connors

NCAR Technical Notes

National Center for
Atmospheric Research
P. O. Box 3000
Boulder, Colorado
80307-3000
www.ucar.edu

NCAR/TN-511+STR

NCAR TECHNICAL NOTES

http://library.ucar.edu/research/publish-technote

The Technical Notes series provides an outlet for a variety of NCAR Manuscripts that contribute in specialized ways to the body of scientific knowledge but that are not yet at a point of a formal journal, monograph or book publication. Reports in this series are issued by the NCAR scientific divisions, serviced by OpenSky and operated through the NCAR Library. Designation symbols for the series include:

EDD – Engineering, Design, or Development Reports
Equipment descriptions, test results, instrumentation, and operating and maintenance manuals.

IA – Instructional Aids
Instruction manuals, bibliographies, film supplements, and other research or instructional aids.

PPR – Program Progress Reports
Field program reports, interim and working reports, survey reports, and plans for experiments.

PROC – Proceedings
Documentation or symposia, colloquia, conferences, workshops, and lectures. (Distribution maybe limited to attendees).

STR – Scientific and Technical Reports
Data compilations, theoretical and numerical investigations, and experimental results.

The National Center for Atmospheric Research (NCAR) is operated by the nonprofit University Corporation for Atmospheric Research (UCAR) under the sponsorship of the National Science Foundation. Any opinions, findings, conclusions, or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

National Center for Atmospheric Research
P. O. Box 3000
Boulder, Colorado 80307-3000

NCAR/TN511+STR
NCAR Technical Note

2014-July

Natural Load Indices (NLI) in NAMD2 load balancing algorithms

Stefan Muszala
Work completed while working at:
1) NCAR, Climate Modeling Section,
Boulder, Colorado
2) Electrical and Computer Engineering,
University of Colorado, Boulder
Presented while working at:
Tech-X Corporation,
Boulder, Colorado
Currently at:
National Center for Atmospheric Research
Terrestrial Sciences
Boulder, Colorado
Gita Alaghband
Computer Science and Engineering,
University of Colorado Denver
James J. Hack
Oak Ridge National Laboratory,
P.O. Box 2008, Oak Ridge, TN
Daniel Connors
Electrical and Computer Engineering,
University of Colorado, Boulder

NCAR Earth System Laboratory
Terrestrial Sciences

NATIONAL CENTER FOR ATMOSPHERIC RESEARCH
P. O. Box 3000
BOULDER, COLORADO 80307-3000
ISSN Print Edition 2153-2397
ISSN Electronic Edition 2153-2400

Table of Contents

List of Figures

List of Tables

Natural Load Indices (NLI) in NAMD2 load balancing algorithms.

Stefan Muszala[1⋆], Gita Alaghband[2], James J. Hack[3], and Daniel Connors[4]

[1] Tech-X Corporation, Boulder CO 80301, USA, muszala@txcorp.com
[2] University of Colorado Denver, Computer Science and Engineering
[3] Oak Ridge National Laboratory P.O. Box 2008, Oak Ridge, TN 37831
[4] University of Colorado, Boulder, Electrical and Computer Engineering

Abstract. Existing measurement-based load indices generally provide adequate performance when used with dynamic load balancing algorithms in parallel scientific applications. An alternative that we present are natural load indices (NLI) that facilitate further performance improvement and better resource usage. Example NLIs are mass of an atom in a Molecular Dynamics (MD) code or rainfall amounts in a climate simulation. This paper presents performance results when we implement NLIs in NAMD2 a MD code. MD simulations are important for drug and medical research and require significant computational resources for long time periods. While an initial cost is incurred during code development in manually determining that NLIs exist, they can then be used at runtime with lower overhead and can reduce total execution time. Results indicate maximum improvement of 21% when comparing existing to NLI based algorithms with 10% overall improvement.

Key words: natural load index, load balancing, NAMD2

1 Introduction

The importance of providing alternative and additional methods to improve and fine-tune performance in parallel scientific applications comes from the increasing complexity and size of those simulations as they strive to model the underlying physics phenomena with more accuracy. One such class of codes are Molecular dynamics (MD) simulations that are vital tools used by medical and pharmaceutical researchers. MD simulations are undergoing changes in which higher atom counts, longer simulations and higher resolutions are increasingly taxing system resources and lengthening the time to completion[1, 2]. Decreasing the

⋆ Support was granted by the Tech-X Corporation and the Climate Modeling Section of the National Center for Atmospheric Research. Computer time was provided by NSF MRI Grant #CNS0421498, NSF MRI Grant #CNS0420873, NSF MRI Grant #CNS0420985, NSF sponsorship of the National Center for Atmospheric Research, the University of Colorado, and a grant from the IBM Shared University Research (SUR) program.

time to completion and reducing the burden of MD codes will save computational resources as well as provide faster access to the final output data saving both time and cost. The load balancing discussed in this paper is in the context of application and data partitioned over a distributed memory multiprocessor system including clusters. This discussion does not include uniform access shared memory MIMD systems.

In implementing load balancing schemes within applications the expertise of the domain scientists are crucial in fine tuning and optimization of the code in order to achieve a desired level of load balancing. In addition, the expertise of the computational scientific programmer is also needed to design and implement efficient code for the underlying architecture. The idea of the natural load index, NLI, is to attempt to bring together the expertise of the domain and computational scientist to achieve an efficient dynamic load balancing method for each application. NLIs currently require an initial analysis to identify the appropriate load index canididates; once an NLI has been identified in the physical properties of an application, that same NLI can be effectively used in different application packages simulating the same physical phenomena. This initial step of NLI identification can be automated and with input from the domain scientist can be made very efficient.

A major advantage of the NLI approach is the ability to dynamically and efficiently monitor the changes in the identified NLI(s) during application execution to determine when a load imbalance occurs and to effect a load balancing step when needed. This is a more efficient way to monitor the load than the measurement-based approach. Run-time measurements require additional instrumentation (and hence introduce overhead) as well reduce the accuracy of load balances. NLIs are computed and needed by the application regardless of load balancing methods. The goal of this paper is to demonstrate the effectiveness of NLI and does not focus on introducing or implementing new load balancing techniques.

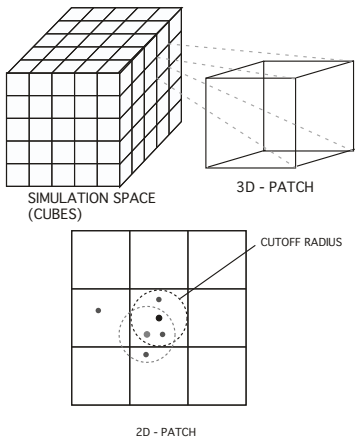

Fig. 1: Diagrams showing the NAMD 3-D computational space (patches) and use of a cutoff radius in a 2-D example.

We present the results and performance comparison of modifying the Molecular Dynamics (MD) code NAMD2 [3, 4] to use *natural* load indices (NLI) as an alternative to the existing NAMD2 *measurement-based* indices [5]. Example NLIs are physical properties such as mass of an atom in a molecular dynamics (MD) model or rainfall amounts in a climate simulation. An NLI is a natural property of or a produced physical quantity of a scientific model that directly or indirectly results in the desired representation of the reality that the model portrays.

This paper offers results in an object oriented MD code written in C++ and shows a more general applicability of our technique. We are able to show improvement without drastically changing the underlying code or existing load balancing algorithms on three differing computer architectures, not only when comparing NLI vs. non-NLI for individual algorithms but NLI also provide the best overall performance when compared to all tested algorithms. These qualities demonstrates applicability to a variety of applications and architectures that use similar algorithms. Our prior work showed how an NLI could be implemented in a grid decomposition, procedural based (Fortran 90) climate model. Those results were in the context of a simulation framework [6].

The remainder of the paper is organized as follows. Section 2 presents related material pertinent to NAMD2 and its load balancing method. Section 3 introduces the characteristics required of an NLI and subsequent correlations with measured timing data. Section 4 describes the experimental framework, implementation and the NAMD2 modifications that were required. Section 4 also presents individual algorithm performance and overall NAMD2 performance. Finally, Section 5 discusses conclusions and directions of future work.

2 NAMD2 Background

The decomposition scheme used in NAMD divides the computation space into a regular number of cubic regions called patches whose size is slightly larger than the cutoff radius (Figure 1). The cutoff radius is the distance beyond which certain forces (van der Walls) are not calculated since the contribution at those distances is minimal. This also has the effect of requiring a patch to interact with only its nearest 26 neighbors [5].

Associated with each patch are a number of moveable and non-moveable compute objects which together are responsible for force computations associated with the atoms assigned to a particular patch [4]. Proxy patches are replicated patches normally residing on remote processors and remove the necessity of communication in certain computations. Moveable compute objects are responsible for the non-bonded force calculations and require the most simulation time while non-migratable work consists of bonded calculations and other background work [7, 5]. The migratable (also moveable) compute objects are those that are redistributed during a load balancing event [7, 5]. Further descriptions of NAMD2 used in this work are included in [8–12].

NAMD2 offers four load balancing algorithms *refine, alg7, orb* and *neighbor* whose combinations offer five configurations that may be set by the user (**refineonly, alg7, orb, other** and **neighbor**). The load balancing portion of NAMD2 is largely undocumented [13], but the implementation may be understood by examining the code itself [14] as well as Kumar et al. [15]. The methods are as follows: *refine* is a refinement procedure, *alg7* is a greedy algorithm, *orb* is orthogonal recursive bisection, *other* calls *alg7* followed by *refine*; all require every processor to communicate with one master processor (they are "Central"

algorithms). In contrast *neighbor* is a distributed load balancer that only balances load with a certain number of nearest processors.

The remainder of the paper will refer only to *refine, alg7, orb* and *neighbor* because we are testing one individual invocation of an algorithm without regard for combinations and duplications (ie. calling *refine* three times in one load balancing invocation or calling *orb* followed by *refine*.

3 Correlation Discovery

Load imbalances are often determined by measuring the execution time of a partition of a model that is assigned to a processor. The measured execution time is what we refer to as timing data which is the load index used in NAMD2 [3, 5]. In order to remove the code related to the timing of a partition and prior to implementing an NLI in the NAMD2 load balancing algorithms we had to be confident that some physical quantity existed in the model that could effectively be used as a load index.

Our earlier work related to CCSM/CAM3 [6] illustrated in detail that correlations between timing data and physical quantities (NLIs) are effective for load balancing. Our experience with CCSM/CAM3 has shown that an NLI can be identified by studying the underlying physics equations that drive the simulation, studying the model code itself or by

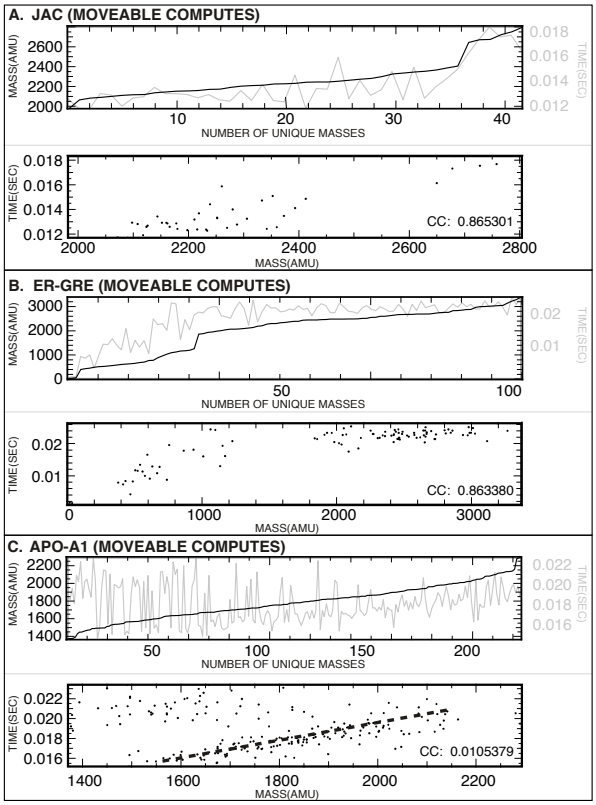

Fig. 2: Top panel for each benchmark shows the values of the individual masses (black lines) as well as the averaged timing data (gray lines) for a uniquely occurring mass. The bottom panel of each set is a scatter plot of uniquely occurring mass versus the averaged time for the computes associated with that mass. CC stands for correlation coefficient. The dashed black line in the bottom-most panel denotes the secondary trend referred to in the text.

comparing a number of physical parameters from the model with timing data. In the case of an MD simulation a logical starting point would be Newton's equation of motion and the observation that mass is the fundamental quantity in force, momentum and acceleration calculations.

Early work by us involving LAMMPS (Large-scale Atomic/Molecular Massively Parallel Simulator) [16, 17] indicated that atomic mass corresponded to processing time although we did not explicitly calculate correlations at that time and none are included in this paper. When we moved to NAMD, we made these correlations explicit. We used LAMMPS in order to show that correlation results from one implementation could be applied to another, NAMD2. The correlations from NAMD2 shown below show that our observations from LAMMPS in which a spatial decomposition scheme is used in initially assigning atoms to processors were indeed readily applicable to NAMD2.

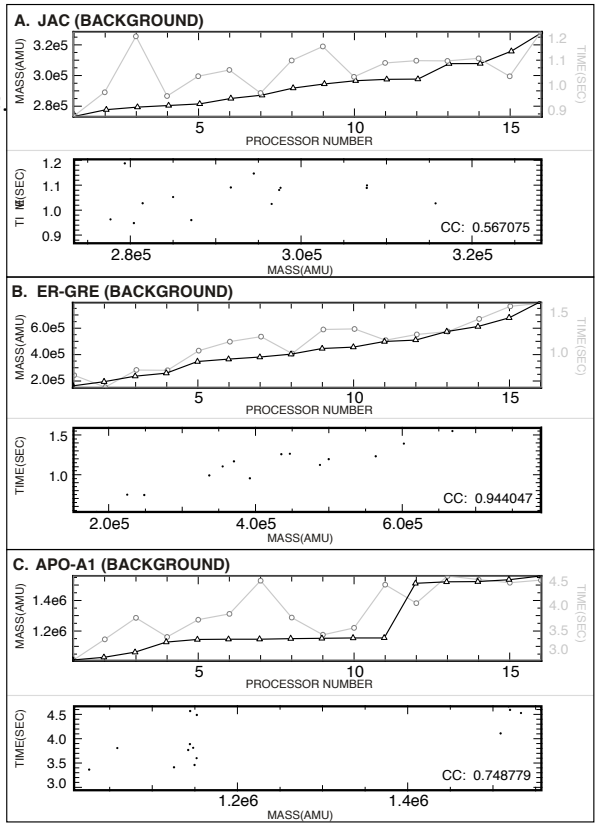

Fig. 3: Top panel for each benchmark shows the values of the masses associated with each processor (black lines with triangles) along with the background timing load (gray lines with circles) for that processor. The bottom panel of each set is a scatter plot of aggregate mass per processor versus time per processor. CC stands for correlation coefficient.

In order for us to use mass as an NLI, the mass of a particular decomposition domain had to correlate with it's respective execution time. Confirming this behavior in NAMD2 consisted of collecting the timing data for each moveable compute object as well as the timed background load for each processor. The timing data from the compute objects was compared directly to the aggregate mass (in atomic mass units (AMU)) of the atoms associated with that compute object. Recall from Section 2 that a moveable compute object

is based on a patch so the aggregate mass assigned to a compute object is actually the aggregate mass of the patch from which it originates. The background load in AMU is calculated by summing the masses from the patches assigned to a specific processor.

The correlations between the timing data and atom mass for the three benchmarks' moveable compute and background loads are shown in Figures 2 and 3. Data processing for the moveable compute objects consisted of calculating the average time for each individual mass present in that particular benchmark. Because timing data varies considerably when measuring complex code in modern computer systems (this due to the effects of cache, interrupts, I/O and operating system events) an average value is a better indicator of long term statistical behavior. Note that the correlation coefficients are generally good (a value of 1.0 indicates perfect correlation, see our work in [6]) with the exception of APO-A1 (Apolipoprotein A-I) moveable computes. Even though the APO-A1 correlations for moveable computes are very low, a load balance using AMU is still possible because: 1) The APO-A1 correlations for background load is sufficiently high and 2) there is a secondary trend that shows very good correlations for higher aggregate masses (dashed line, bottom panel of Figure 2C).

4 Experimental Approach

Experiments were carried out for each of three MD benchmarks, JAC (the protein dihydrofolate reductase), ER-GRE (an estrogen receptor) and APO-A1 because of their accessibility and usage in the scientific literature and because they are indicative of problems currently being simulated with MD codes. We tested the four NAMD2 load balancing algorithms: *refine, alg7, orb* and *neighbor*, abbreviated *neigh* [15]. Table 2 shows processor counts as well as the results of comparing a particular algorithm's speedup using the m -based implementation and the modified NLI version. An important note regarding scaling is that as processor counts increase for a specific simulation and load balancing

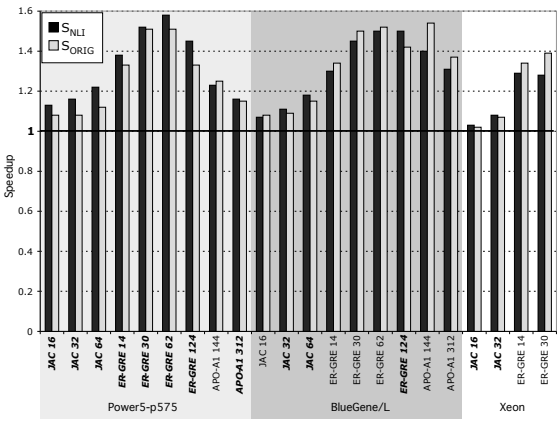

Fig. 4: Bar graph showing NLI and ORIG speedups (compared to no load balancing) for each processor count, simulation type and platform. Cases in which NLI outperformed the original implementation are shown in italic-bold typeface. ORIG refers to the original implementation in NAMD before we applied any modifications.

method that the NLI technique improves (Table 2). Figure 4 shows the overall best performance for the original and NLI implementations graphically.

The results of the new version of NAMD2 using an NLI as the load indicator for load balancing shows improvement in 65 out of 88 trials in which individual performance of an algorithm was compared. The best improvement for an individual algorithm was 21%. Furthermore 13 out of 22 trials indicate that using an NLI gives the best overall performance for a given simulation (10%). Experiments were carried out on an IBM Power5, IBM BlueGene/L and Intel Xeon systems. It is important to note that the trials for which better performance was not obtained were still within very close proximity of the measurement-based index. In this work we did not attempt to design and optimize load balancing techniques for specific use with an NLI.

Table 1: Standard deviations calculated from wall-clock time per time-step for two simulations on the Power5-p575 (Data corresponds to simulation in Figure 5) and BlueGene/L platforms before and after load balancing for NLI and the original NAMD2 implementation.

		Standard Deviation	
Platform		ORIG	NLI
Power5-p575	Total Simulation	0.00368	0.00286
	After Load Balance	0.00213	0.00193
BlueGene/L	Total Simulation	0.01316	0.01004
	After Load Balance	0.01008	0.00874

Speedups reported are over already load-balanced code and while the speedups presented are small, they add up to large absolute time savings in long MD simulations. Consider that the best case scenario from the ER-GRE benchmark at 124 processors on the IBM Power5 decreases the execution time per time-step from 0.0152 to 0.0138 seconds (Δ0.0014 sec.). That value can amount to an absolute time savings of over 38 (out of 422) hours during a 10^8 time-step simulation. Similar arguments have successfully been applied to climate models [6] and are applicable to other large simulations such as those found in the fusion, ocean modeling and combustion modeling communities. Keep in mind that we are applying NLIs to already highly-optimized codes with existing mature load balancing systems.

The wall-clock time per individual time-step indicates that NLI produces data with less variation and hence lower standard deviations than that produced by the equivalent measurement-based simulation. Table 1 shows data from two parallel platforms with standard deviations calculated after the load balance and calculated for the entire simulation. Figure 5 illustrates the wall-clock time per time-step data from which the IBM Power5-p575 standard deviations in Table 1

are calculated. A prominent variation between NLI (shown in Figure 5) and the original method is that NLI allows a load balance to be invoked at the start of the simulation and removes the necessity for collecting the statistics required of a measurement-based method; NLI to this point has a head start. The behavior of the wall-clock time per time-step after a load balance has been invoked further indicates that NLI outperforms the original method because of the lower values of the calculated standard deviations (Table 1) which indicates a more accurate distribution of work.

5 Conclusions and Future Work

The results demonstrated are encouraging for a number of reasons. We are able to apply more efficient load balances due to the removal of timer infrastructure and overhead, because we can apply load balances earlier in a simulation and because we are making more accurate load balancing decisions. Detailed points include: 1) In the work regarding CCSM/CAM3 we were able to demonstrate how an NLI could be implemented in a grid decomposition, procedural based (Fortran90) model using a marker for moist convective rainout [6]. Those results were in the context of a simulation framework. This NLI implementation offers results in an object oriented MD code written in C++ and shows a more general applicability of our technique. 2) When modifying NAMD2, we did not want to make drastic changes to the underlying structure of the code in order to show that the NLI method was effective and

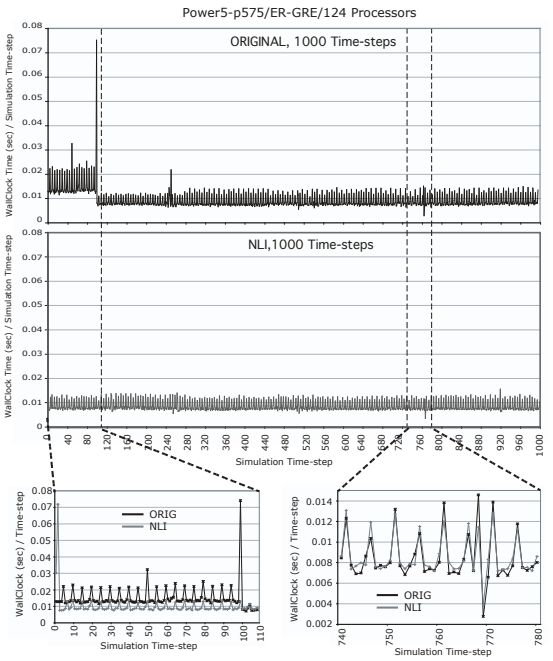

Fig. 5: Wall-clock time /simulation time-step plotted for each of 1000 time-steps in a 124 processor ER-GRE simulation on the IBM Power5-p575. Black lines represent the measurement-based original NAMD2 load balancing implementation; Grey lines represent the same load balancing technique using NLIs. Note that fewer spurious timing events occur with NLI. The break at time-step 100 is where the load balancing occurs for the original implementation and is required since timing statistics must be collected. NLI allows a load balance to be executed at time-step 2.

straightforward to implement. 3) Using existing algorithms shows that using an NLI is advantageous without changing anything in the algorithm proper. This demonstrates applicability to a variety of applications that may use similar algorithms. 4) Overall improvement occurs when using NLI in NAMD2 particularly on the IBM Power5 platform. As processor count increases, NLI performance generally increases. 5) Correlations and characteristics from one benchmark do not drastically change when executing them on different architectures. All correlations were collected from 1 processor simulations executed on the Intel Xeon cluster, but performance improvement spanned all architectures. 6) The NLI gives application developers another option for making load balancing decisions which in certain cases has clear advantages over traditional measurement-based load indices.

Current work involves providing a formalization for using the NLI technique in a number of models. Current work also involves a successful implementation in CSSM/CAM3 as well as correlations from POP3 [18], an ocean model and LAMMPS indicate that the NLI technique has a general applicability and should be classified with existing load indices. Future work should investigate specific algorithms that may be more conducive to using the NLI in NAMD2 load balancing implementations. Furthermore, our implementation may benefit from more substantial modifications of NAMD2 as would taking into consideration the masses associated with proxy patches. We also plan to apply load balancing decisions based on atom types and bond types rather than mass alone. Atom type is indicative of the number of available bonds with more bonds requiring more computation. In turn, the bond types themselves may also be indicative of computational differences.

Table 2: Matrix of individual algorithm performance calculated as the time of the original NAMD2 measurement-based execution time divided by the time for the NLI version. Speedups greater than one indicate that our NLI method outperformed the original NAMD2 implementation. Boldface type indicates speedups greater than or equal to 1.0 and shows where an NLI based algorithm is the best performer.

		Power5-p575				BlueGene/L				P4-Xeon			
Sim.	Procs.	*neigh*	*alg7*	*orb*	*refine*	*neighr*	*alg7*	*orb*	*refine*	*neigh*	*alg7*	*orb*	*refine*
jac	16	**1.07**	0.98	**1.06**	**1.05**	**1.02**	0.99	**1.02**	0.94	**1.01**	0.99	**1.01**	**1.01**
	32	**1.09**	**1.01**	**1.09**	**1.08**	**1.03**	0.94	**1.06**	**1.02**	**1.01**	0.92	**1.00**	**1.01**
	64	**1.12**	**1.01**	**1.21**	**1.11**	**1.05**	0.98	**1.12**	**1.03**	-	-	-	-
er-gre	14	**1.08**	**1.04**	**1.05**	**1.03**	**1.00**	**1.05**	0.98	0.96	**1.03**	**1.03**	0.98	0.96
	30	**1.08**	**1.02**	**1.01**	0.97	**1.01**	**1.06**	0.98	0.92	**1.04**	**1.04**	**1.06**	0.92
	62	**1.09**	**1.10**	**1.10**	**1.05**	**1.01**	**1.03**	**1.04**	0.98	-	-	-	-
	124	**1.10**	**1.02**	**1.08**	**1.10**	**1.03**	**1.03**	**1.03**	**1.06**	-	-	-	-
apo-a1	144	**1.03**	0.80	**1.05**	0.99	**1.00**	0.89	**1.02**	0.91	-	-	-	-
	312	**1.03**	0.93	**1.06**	**1.00**	**1.00**	0.88	**1.09**	0.96	-	-	-	-

References

1. T. Schlick, R. Skeel, A. Brünger, L. Kalé, J. A.Board Jr, J. Hermans, and K. Schulten, "Algorithmic chanllenges in computational molecular biophysics," *Journal of Computational Physics*, vol. 151, pp. 9–48, 1999.
2. F. Suits, M. C. Pitman, J. W. Pitera, W. C. Swope, and R. S. Germain, "Overview of molecular dynamics techniques and early scientific results from the Blue Gene project," *IBM Journal of Research and Development*, vol. 49, no. 2/3, 2005.
3. J. C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. D. Skeel, L. Kal, and K. Schulten, "Scalable molecular dynamics with namd.," *J Comput Chem*, vol. 26, pp. 1781–1802, December 2005.
4. L. Kalé;, R. Skeel, M. Bhandarkar, R. Brunner, A. Gursoy, N. Krawetz, J. Phillips, A. Shinozaki, K. Varadarajan, and K. Schulten, "Namd2: Greater scalability for parallel molecular dynamics," *J. Comput. Phys.*, vol. 151, no. 1, pp. 283–312, 1999.
5. J. Phillips, G. Zheng, and L. V. Kalé, "Namd: Biomolecular simulation on thousands of processors," in *Workshop: Scaling to New Heights*, (Pittsburgh, PA), May 2002.
6. S. P. Muszala, J. J. Hack, D. A. Connors, and G. Alaghband, "The Promise of Load Balancing the Parameterization of Moist Convection Using a Model Data Load Index," *Journal of Atmospheric and Oceanic Technology*, vol. 23, pp. 525–537, 2006.
7. R. K. Brunner, "Versatile automatic load balancing with migratable objects," TR 00-01, January 2000.
8. R. K. Brunner and L. V. Kalé, "Handling application-induced load imbalance using parallel objects," in *Parallel and Distributed Computing for Symbolic and Irregular Applications*, pp. 167–181, World Scientific Publishing, 2000.
9. L. V. Kalé, M. Bhandarkar, and R. Brunner, "Load balancing in parallel molecular dynamics," in *Fifth International Symposium on Solving Irregularly Structured Problems in Parallel*, vol. 1457 of *Lecture Notes in Computer Science*, pp. 251–261, 1998.
10. "Namd users guide: http://www.ks.uiuc.edu/ research/ namd/ 2.6/ ug/ node54.html."
11. R. Brunner, A. Dalke, A. Gursoy, W. Humphrey, and M. Nelson, "NAMD Programming Guide," tech. rep., Theoretical Biophysics Group, University of Illinois and Beckman Institute, Urbana, IL 61801, USA., 1998.
12. G. Zheng, *Achieving High Performance on Extremely Large Parallel Machines: Performance Prediction and Load Balancing*. PhD thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, 2005.
13. "Namd mailing list: http://www.ks.uiuc.edu/ research/ namdmailing_list/ namdl/ 1502.html."
14. "Namd code resource: http://www.ks.uiuc.edu/ research/ namd/ doxygen/ index.html."
15. S. Kumar, C. Huang, G. Almasi, and L. V. Kalé, "Achieving strong scaling with NAMD on Blue Gene/L," in *Proceedings of IEEE International Parallel and Distributed Processing Symposium 2006*, April 2006.
16. S. Plimpton, "Fast parallel algorithms for short-range molecular dynamics," *J. Comput. Phys.*, vol. 117, no. 1, pp. 1–19, 1995.
17. S. Plimpton, R. Pollock, and M. Stevens, "Particle-mesh ewald and rrespa for parallel molecular dynamics simulations.," in *PPSC*, 1997.
18. D. J. Kerbyson and P. W. Jones, "A Performance Model of the Parallel Ocean Program," *Int. J. High Performance Computing Applications*, 2005.