NAMD: Biomolecular Simulation on Thousands of Processors[*]

James Phillips[†] Gengbin Zheng[‡] Laxmikant Kalé[§]

Abstract

NAMD is a parallel, object-oriented molecular dynamics program designed for high performance simulation of large biomolecular systems. NAMD employs the prioritized message-driven execution capabilities of the Charm++/Converse parallel runtime system, allowing excellent parallel scaling on both massively parallel supercomputers and commodity workstation clusters. This paper discusses the techniques which have allowed NAMD to effectively employ over one thousand processors in production simulations of biomedical relevance.

1 Introduction

NAMD is a parallel, object-oriented molecular dynamics program designed for high performance simulation of large biomolecular systems [7]. NAMD employs the prioritized message-driven execution capabilities of the Charm++/Converse parallel runtime system,[1] allowing excellent parallel scaling on both massively parallel supercomputers and commodity workstation clusters. NAMD is distributed free of charge via the web[2] to over 3000 registered users as both source code and convenient precompiled binaries. NAMD development and support is a service of the National Institutes of Health Resource for Macromolecular Modeling and Bioinformatics, located at the University of Illinois at Urbana-Champaign.[3]

Molecular dynamics simulation. In a molecular dynamics (MD) simulation, full atomic coordinates of the proteins, nucleic acids, and/or lipids of interest, as well as explicit water and ions, are obtained from known crystallographic or other structures. An empirical energy function, which consists of approximations of covalent interactions in addition to long-range Lennard-Jones and electrostatic terms, is applied. The resulting Newtonian equations of motion are typically integrated by symplectic and reversible methods using a timestep of 1 fs. Modifications are made to the equations of motion to control temperature and pressure during the simulation.

The basic protocol for MD simulations consists of minimization (to eliminate initial contacts which would destabilize the integrator), equilibration to a temperature of 300 K and a pressure of 1 atm, and simulation in an isobaric (NPT) ensemble for 1-10 ns. This is sufficient to test the stability of a biomolecular aggregate or to observe the relaxation of the system into a more favorable

[*] This work was supported by the National Institutes of Health (NIH PHS 5 P41 RR05969-04) and the National Science Foundation (NSF/GCAG BIR 93-18159 and NSF BIR 94-23827 EQ).

[†] Beckman Institute, University of Illinois, Urbana, IL.

[‡] Department of Computer Science and Beckman Institute, University of Illinois, Urbana, IL.

[§] Department of Computer Science and Beckman Institute, University of Illinois, Urbana, IL.

[1] `http://charm.cs.uiuc.edu/`

[2] `http://www.ks.uiuc.edu/Research/namd/`

[3] `http://www.ks.uiuc.edu/`

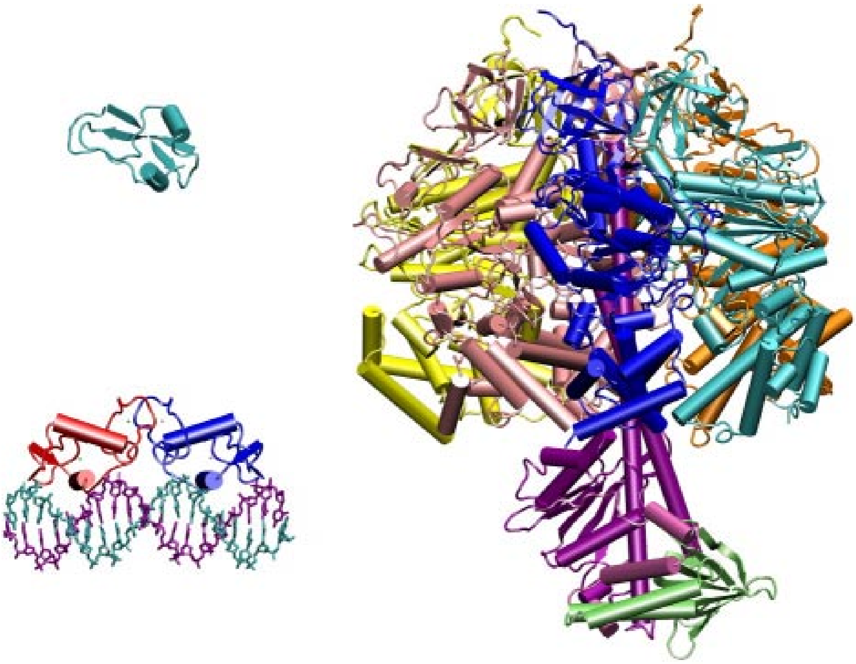

Figure 1: Simulations have increased exponentially in size, from BPTI (upper left, about 3K atoms), through the estrogen receptor (lower left, 36K atoms, 1996), to F_1-ATPase (right, 327K atoms, 2001). (Atom counts include solvent.)

conformation. However, if the goal is to study events which would not spontaneously occur during the timespan of the simulation, the system may be forced to undergo a transition via the application of forces in a steered molecular dynamics protocol. Important observations may be made during such a simulation of non-equilibrium events, even though the simulated time scale is much shorter than the natural one.

Importance of molecular dynamics simulation software. The application of molecular dynamics simulation methods in biomedicine is directly dependent upon the capability, performance, usability and availability of the required simulation and analysis software. Simulations often require substantial computing resources, sometimes available only at supercomputing centers. By developing the program NAMD, the Resource provides the biomedical research community with a freely available tool for performing high quality MD simulations of large biomolecular systems using a variety of available and cost-effective hardware [7, 10].

Increases in system size and simulation length. With continuing increases in high performance computing technology, the domain of biomolecular simulation has rapidly expanded from isolated proteins in solvent to include complex aggregates, often in a lipid environment. Such simulations can easily exceed 100,000 atoms (see Fig. 1). Similarly, studying the function of even the simplest of biomolecular machines requires simulations of 10 ns or longer, even when techniques for accelerating processes of interest are employed. The goal of interactive simulation of smaller molecules places even greater demands on the performance of MD software, as the sensitivity of the haptic interface increases quadratically with simulation speed [14].

Importance of parallel computing to simulations. Despite the seemingly unending progress in microprocessor performance indicated by Moore's law, the urgent nature and computational needs of biomedical research demand that we pursue the additional factors of tens, hundreds, or thousands in total performance which may be had by harnessing a multitude of processors for a single calculation. While the MD algorithm is blessed with a large ratio of calculation to data, its parallelization to large numbers of processors is not straightforward [4]. The Resource has been particularly prescient in recognizing and pursuing the benefits of parallel computing for biomedical research.

Increasing availability of large parallel resources. The Accelerated Strategic Computing Initiative,[4] a U.S. Department of Energy program, has provided an unprecedented impetus for the application of massively parallel teraflop computing to the problems of the physical sciences and engineering. The National Science Foundation has followed this lead, funding terascale facilities at the national centers with the intent of enabling research that can employ many or all of the processors on these new machines.[5] The concept of grid computing promises to make this computational power readily available from any desktop. Huge computing resources will soon be available to the biomedical researcher who is able to harness it using software such as NAMD.

Other available simulation programs. The biomolecular modeling community sustains a variety of software packages with overlapping core functionality but varying strengths and motivations. For comparison, we select AMBER, CHARMM, GROMACS, NWChem, and TINKER.

AMBER [15] and CHARMM [3] are often considered the standard "community codes" of structural biology, having been developed over many years by a wide variety of researchers. Both AMBER and CHARMM support their own force field development efforts, although the form of the energy functions themselves is quite similar. Both codes are implemented in FORTRAN 77, although AMBER takes the form of a large package of specialized programs while CHARMM is a single binary. The parallelization of these codes is limited and not uniform across features, e.g., the GIBBS module of AMBER is limited to costly shared memory machines. Neither program is freely available, although the academic versions are highly discounted in comparison to commercial licenses.

GROMACS 3.0 [9], the version recently released, claims the title of "fastest MD." This can be attributed largely to the GROMOS force field, which neglects most hydrogen atoms and eliminates van der Waals interactions for those that remain. In contrast, the AMBER and CHARMM force fields represent all atoms and new development has centered on increasing accuracy via additional terms. Additional performance on Intel x86 processors comes from the implementation of inner loops in assembly code. GROMACS is implemented in C as a large package of programs and is newly released under the GNU General Public License (GPL). Distribution takes the form of source code and Linux binaries. Plans for future support and development are unclear for this program.

NWChem [6] is a comprehensive molecular simulation system developed by a large group of researchers at the PNNL EMSL, primarily to meet internal requirements. The code centers on quantum mechanical methods but includes an MD component. Implemented in C and FORTRAN,

[4] `http://www.llnl.gov/asci/`

[5] `http://www.interact.nsf.gov/cise/descriptions.nsf/pd/tcs/`

NWChem is parallelized using MPI and a Global Arrays library[6] which automatically redistributes data on distributed memory machines. Parallel scaling is respectable given sufficient workload, although published benchmarks tend to use abnormally large cutoffs rather than the 12 Å (or PME) typically used in biomolecular simulations. Access to NWChem source code is available with the submission of a signed license agreement, although support is explicitly unavailable outside of PNNL.

TINKER [12] is a small FORTRAN code developed primarily for the testing of new methods. It incorporates a variety of force fields, in addition to its own, and includes many experimental methods. The code is freely available, but is not parallelized, and is therefore inappropriate for traditional large-scale biomolecular simulations. It does, however, provide the community with a simple code for experiments in method development.

Motivation for NAMD development. NAMD builds upon other available programs in the field by incorporating popular force fields and reading file formats from other codes. NAMD complements existing capabilities by providing a higher performance alternative for simulations on the full range of available parallel platforms. Through NAMD, the Resource expands the MD user community by providing free, ready-to-run, easy-to-use software for most available platforms, complete with web-based educational materials and a safety net of support via email. NAMD also provides easily modified object-oriented C++ source code to support method development both within and beyond the Resource.

2 NAMD Parallelization Strategy

We have approached the scalability challenge by adopting message-driven approaches and reducing the complexity associated with these methods by combining multithreading and an object-oriented implementation in C++.

Charm++ parallel object-based programming model. The dynamic components of NAMD are implemented in the Charm++[8] parallel language. Charm++ implements an object-based message-driven execution model. In Charm++ applications, there are collections of C++ objects, which communicate by remotely invoking methods on other objects by messages.

Compared with conventional programming models such as message passing, shared memory or data parallel programming, Charm++ has several advantages in improving locality, parallelism and load balance. The flexibility provided by Charm++ is a key to the high performance achieved by NAMD on thousands of processors.

In Charm++ applications, users decompose the problem into objects, and since they decide the granularity of the objects, it is easier for them to generate parallelism. As described in the paragraph below, NAMD uses a novel way of decomposition that easily generates the large amount of parallelism needed to occupy thousands of processors.

Charm++'s object-based decomposition also help users to improve data locality. Objects encapsulate states, Charm++ objects are only allowed to directly access their own local memory. Access to other data is only possible via asynchronous method invocation to other objects.

Charm++'s parallel objects and data-driven execution adaptively overlaps communication and computation and hide communication latency: when an object is waiting for some incoming data, entry functions of other objects with all data ready are free to execute.

[6] `http://www.emsl.pnl.gov:2080/docs/global/ga.html`

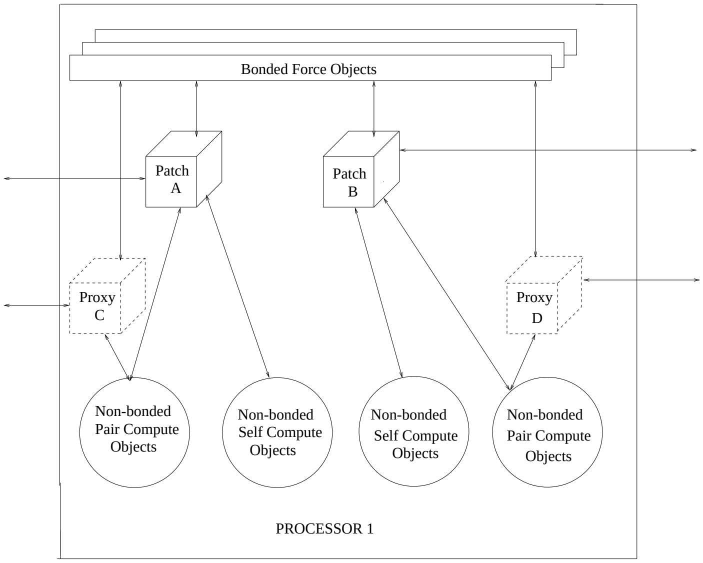
Bonded Force Objects
Patch A
Patch B
Proxy C
Proxy D
Non-bonded Pair Compute Objects
Non-bonded Self Compute Objects
Non-bonded Self Compute Objects
Non-bonded Pair Compute Objects
PROCESSOR 1

Figure 2: NAMD 2 hybrid force/spatial decomposition.

In Charm++, objects may even migrate from processor to processor at runtime. Object migration is controlled by Charm++ load balancer. Charm++ implements a measurement based load balancing framework which automatically instruments all Charm++ objects, collectes computation load and communication pattern during execution and stores them into a ”load balancing database”. Charm++ then provides a collection of load balancing strategies whose job is to decide on a new mapping of objects to processors based on information from the database. Load balancing strategies are implemented in Charm++ as libraries. Programmers can easily experiment with different existing strategies by linking different strategy modules and specify which strategy to use at runtime via command line options. This involves very little efforts from programmers while achieving significant improvements in performance in adaptive applications. Application specific load balancing stategies can also be developed by users and plugged in easily. In the following paragraphs, we will describe the load balancing strategies optimized for NAMD2 in details.

Hybrid force/spatial decomposition. NAMD 1 is parallelized via a form of spatial decomposition using cubes whose dimensions are slightly larger than the cutoff radius. Thus, atoms in one cube need to interact only with their 26 neighboring cubes. However, one problem with this spatial decomposition is that the number of cubes is limited by the simulation space. Even on a relatively large molecular system, such as the 92,442 atom ApoA-I benchmark, we only have 245 $(7 \times 7 \times 5)$ cubes. Further, as density of the system varies across space, one may encounter strong load imbalances.

NAMD 2 addresses this problem with a novel combination of force [11] and spatial decomposition. For each pair of neighboring cubes, we assign a non-bonded force computation object, which can be independently mapped to any processor. The number of such objects is therefore 14 times (26/2 + 1 self-interaction) the number of cubes.

The cubes described above are represented in NAMD 2 by objects called *home patches*. Each home patch is responsible for distributing coordinate data, retrieving forces, and integrating the equations of motion for all of the atoms in the cube of space owned by the patch. The forces used by the patches are computed by a variety of *compute objects*. There are several varieties of compute objects, responsible for computing the different types of forces (bond, electrostatic, constraint, etc.). Some compute objects require data from one patch, and only calculate interactions between atoms within that single patch. Other compute objects are responsible for interactions between atoms distributed among neighboring patches. Relationships among objects are illustrated in Fig. 2.

Dynamic communication and execution. When running in parallel, some compute objects require data from patches not on the compute object's processor. In this case, a *proxy patch* takes the place of the home patch on the compute object's processor. During each time step, the home patch requests new forces from local compute objects, and sends its atom positions to all its proxy patches. Each proxy patch informs the compute objects on the proxy patch's processor that new forces must be calculated. When the compute objects provide the forces to the proxy, the proxy returns the data to the home patch, which combines all incoming forces before integrating. Thus, all computation and communication is scheduled based on priority and the availability of required data.

Measurement-based load balancing. Some compute objects are permanently placed on processors at the start of the simulation, but others are moved during periodic load balancing phases. Ideally, all compute objects would be able to be moved around at any time. However, where calculations must be performed for atoms in several patches, it is more efficient to assume that some compute objects will not move during the course of the simulation. In general, the bulk of the computational load is represented by the non-bonded (electrostatic and van der Waals) interactions, and certain types of bonds. These objects are designed to be able to migrate during the simulation to optimize parallel efficiency. The non-migratable objects, including computations for bonds spanning multiple patches, represent only a small fraction of the work, so good load balance can be achieved without making them migratable.

NAMD uses a measurement-based load balancer, employing the Charm++ load balancing framework. When a simulation begins, patches are distributed according to a recursive coordinate bisection scheme [2], so that each processor receives a number of neighboring patches. All compute objects are then distributed to a processor owning at least one home patch, insuring that each patch has at most seven proxies. The dynamic load balancer uses the load measurement capabilities of Converse to refine the initial distribution. The framework measures the execution time of each compute object (the object loads), and records other (non-migratable) patch work as "background load." After the simulation runs for several timesteps (typically several seconds to several minutes), the program suspends the simulation to trigger the initial load balancing. NAMD retrieves the object times and background load from the framework, computes an improved load distribution, and redistributes the migratable compute objects.

Success on large supercomputers. NAMD's combination of force and spatial decomposition allows large simulations to be efficiently decomposed onto hundreds of processors. Our success in employing this technique is demonstrated by a parallel speedup of 1252 on 2048 CPUs of the Sandia ASCI Red for a 206,617 atom cutoff (non-PME) simulation [4] and by our selection as a finalist for the 2000 Gordon Bell prize. When combined with ongoing work on improving the parallelization of PME simulations, NAMD is uniquely positioned as the tool of choice for simulations employing up to a thousand processors on the new generation of terascale platforms, i.e., the 3,000

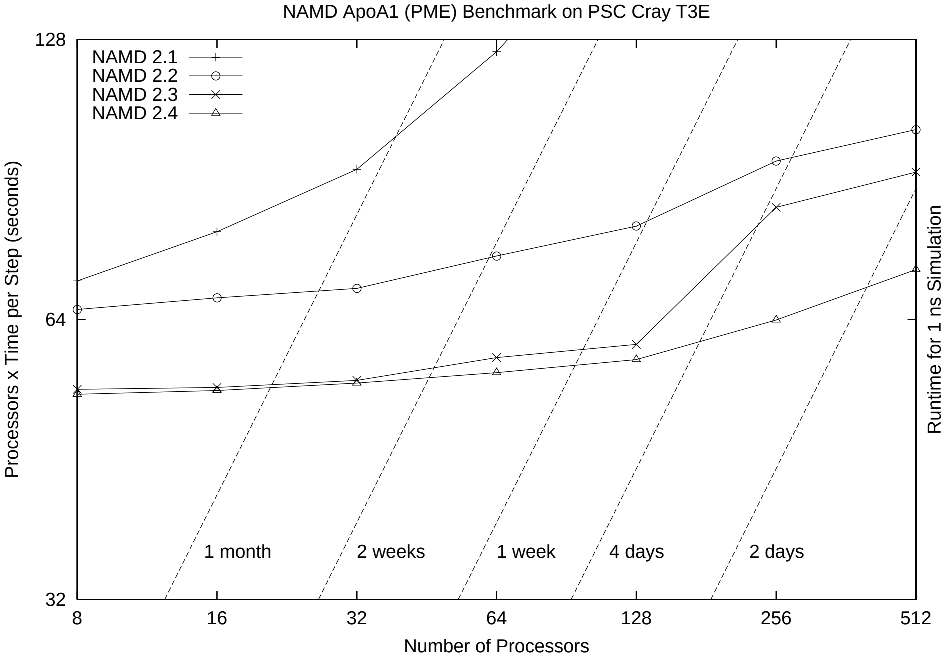

Figure 3: Total resources consumed per step for 92K atom benchmark with and without particle–mesh Ewald by NAMD versions 2.1–2.3 on varying numbers of processors of the PSC T3E. Perfect linear scaling is a horizontal line. Diagonal scale shows runtime per ns. Note that performance increases for both PME and cutoff simulations with each release.

processor AlphaServer SC and the 1,000 processor Itanium cluster at the Pittsburgh and Illinois supercomputing centers.

3 Particle Mesh Ewald in NAMD

Incorporating, tuning, and parallelizing particle–mesh Ewald (PME) [5] full electrostatics calculations has been a recurring theme in NAMD 2 development and this one feature has primarily determined the performance and scalability observed by users. As seen in Fig. 3, significant progress has been made. Development started with the incorporation of the external DPME[7] package into NAMD 2.0, providing a stable base functionality. The PME reciprocal sum was serialized in this version because the target workstation clusters for which DPME was developed obtained sufficient scaling by only distributing the direct interactions. In NAMD, the direct interactions were incorporated into the existing parallel cutoff non-bonded calculation. The reciprocal sum was reimplemented more efficiently in NAMD 2.1, providing a base for later parallelization.

An effective parallel implementation of the PME reciprocal sum was finally achieved in NAMD 2.2. The final design was elegant but far from obvious, as numerous false starts were attempted along the way. In particular, the parallel implementation of FFTW[8] was found to be inappropriate for our purposes so a new one was written for NAMD. The reciprocal sum is currently parallelized only to the size of the PME grid, which is typically between 50 and 128. However, this is sufficient to allow

[7] Distributed Particle–Mesh Ewald, `http://www.ee.duke.edu/Research/SciComp/SciComp.html`

[8] Fastest Fourier Transform in the West, `http://www.fftw.org/`

NAMD simulations to scale to several hundred processors as the bottleneck has been significantly reduced. In addition, the message-driven design of NAMD interleaves the PME communication with other work, giving good performance even on high latency networks.

Improvements to the PME direct sum in NAMD 2.3 were obtained by eliminating expensive calls to `erfc()`.[9] This was accomplished by incorporating the entire short-range non-bonded potential into an interpolation table. By interpolating based on the square of the interaction distance, the calculation of $1/\sqrt{r^2}$ was eliminated as well. The interpolation table is carefully constructed to avoid cancellation errors and is iteratively refined during program startup to eliminate discontinuities in the calculated forces. Simulations performed with the new code are up to 50% faster than before and are of equivalent accuracy.

Shared memory optimizations. Although NAMD 2 was initially designed to employ multiple threads on multiprocessor shared-memory machines, particularly in combination with message-passing on a cluster of SMP nodes, this was never implemented. Even on modern clusters with SMP nodes, standard MPI-based message passing is normally the encouraged method of programming. However, it is certainly possible to exploit superior communication speeds between processors on the same node through algorithm design in a single-threaded implementation. Modifications to the NAMD load balancing system and proxy communication algorithms will reduce transmission of coordinates and forces between nodes.

Distributing PME calculations to at most one processor per node for large machines in NAMD 2.4 has reduced contention for available interconnect bandwidth and other switch or network interface resources. This has increase the scalability of NAMD 2 simulations using PME to 1024 processors on the existing machines at NCSA, PSC, and SDSC with 2, 4, and 8 processors per node.

4 Scalability Results

Figure 4 illustrates the portable scalability of NAMD on a variety of recent platforms employed for production simulations by researchers at the Resource. Figure 5 shows the greatest scalability attained by NAMD on the 3000 processor LeMieux cluster at PSC. NAMD scales particularly well for larger simulations, which are those for which improved performance is most greatly desired by researchers.

5 Remaining Challenges

Continuation after external failures. The increasing size and complexity of parallel computing platforms will soon reduce the mean time between failures for hardware and system software to the point that large jobs will be more likely to fail due to an external error than to complete normally. Application software must address this problem internally, as general methods which may be developed are likely to be incomplete or inflexible. While NAMD 2 periodically saves atomic coordinates and velocities, the user is required to alter input files and restart the calculation manually. NAMD 3 will provide a single comprehensive restart file which will save all relevant information, including the state of the command interpreter. This will allow a simulation which has ended prematurely to be continued automatically. In a second, more aggressive approach, we

[9] The erfc(x) function computes the complement of the error function of x.

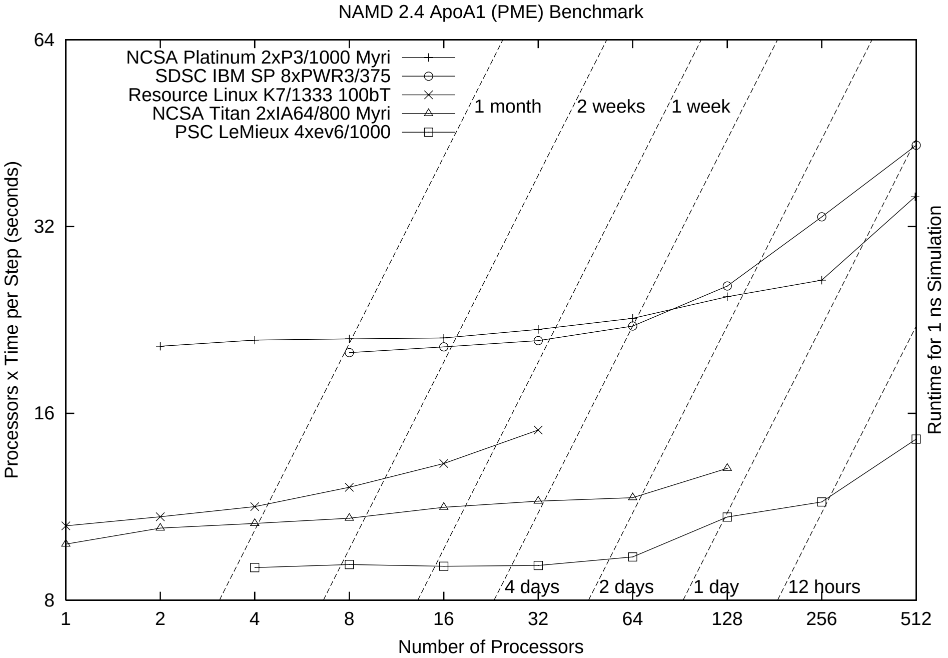

Figure 4: Total resources consumed per step for 92K atom benchmark with particle–mesh Ewald by NAMD 2.4 on varying numbers of processors for recent parallel platforms. Perfect linear scaling is a horizontal line. Diagonal scale shows runtime per ns.

will develop methods to allow a running simulation to survive and rapidly recover from the loss of a small number of nodes, dynamically reallocating calculations to the remaining processors.

Repeatable and deterministic parallel simulations. In NAMD 2, the order of operations, particularly the summation of forces, is determined dynamically by the load balancer and the order of message arrival. Combined with the non-associativity of floating-point arithmetic and the chaotic nature of MD simulations, this causes parallel simulations starting from identical initial conditions to diverge after a few hundred steps, even when run on the same machine and number of processors. In order to obtain the benefits of exact repeatability for both the developer (needed to perform long running regression tests) and for the user (repeating a simulation in order to extract additional details), NAMD 3 will optionally employ associative simulated fixed-point addition for force summation. This can be done at almost no additional cost [13].

Accelerating interactive simulation. Interactive simulation will become truly useful when a researcher can obtain 1 ps of simulation time per second of wall clock time. At this rate 1 ns is simulated in 20 minutes and 1 μs in under two weeks. Obtaining this level of performance, which is at least a factor of ten beyond current performance on any platform, will require the distribution of a simulation of 10,000 atoms efficiently to 256 processors of a high performance, tightly coupled machine. Latency-tolerant minimally coupled algorithms will be essential in this environment, as any synchronization would harm performance. Force computations must be partitioned into even smaller, fine-grained, objects than currently attained. In addition to the parallelization challenges, careful compromises between performance and accuracy will need to be made in selecting algorithms

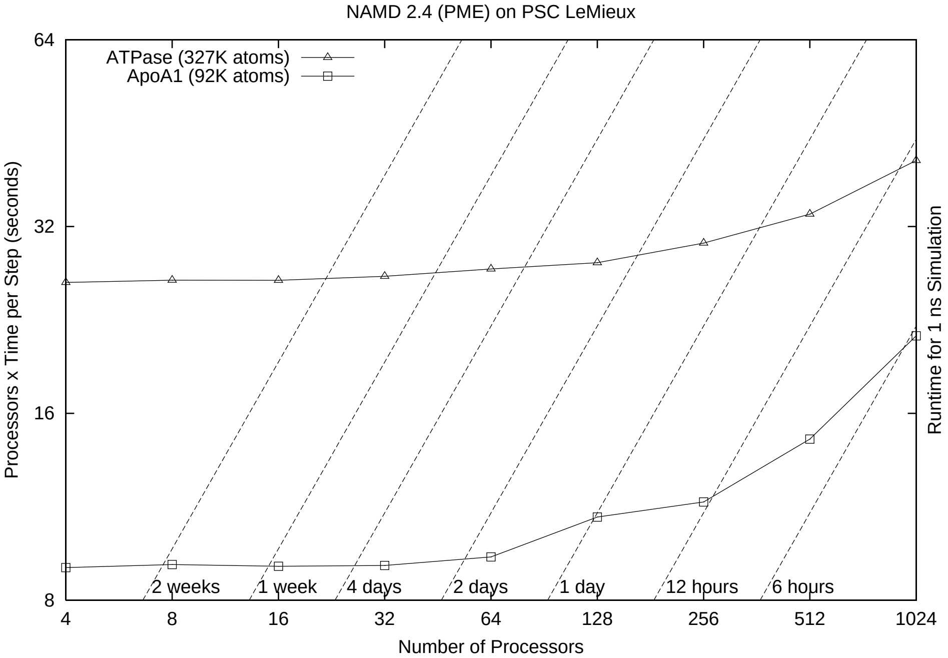

Figure 5: Total resources consumed per step for 92K and 327K atom benchmarks with particle–mesh Ewald by NAMD 2.4 on varying numbers of processors for PSC LeMieux. Perfect linear scaling is a horizontal line. Diagonal scale shows runtime per ns.

for both force calculation and integration.

Large capability clusters. All recent terascale systems have combined standard multiprocessor workstations with proprietary low latency interconnects. Common examples are the IBM SP (San Diego),[10] AlphaServer SC (Pittsburgh),[11] and Myrinet Linux clusters (Illinois).[12] In order to scale to thousands of processors on these platforms, NAMD will have to take advantage (through Converse) of the low level interfaces to the interconnect hardware (e.g., IBM's LAPI [1]) which translate to a message driven design much better than the common MPI interface. Multiprocessor nodes are used on these machines (up to 32 CPUs per node on ASCI Q, the AlphaServer SC being installed at Los Alamos[13]), so NAMD will be adapted to this multilevel communication hierarchy. This will be in the form of multiple threads on some platforms while merely modifying communication patterns on others. In either case, load balancing will need to consider the disparity of communication speeds between processors.

Massively parallel platforms. Supercomputers with over 1000 processors are available at the national centers (San Diego, Pittsburgh, and Illinois) and machines with over 10,000 processors are in use at DOE labs. We expect computational facilities with 10,000 to 100,000 processors to be available for simulations by the middle of the proposed funding period, e.g., IBM is expected to build

[10] http://www.sdsc.edu/Resources/bluehorizon.html

[11] http://www.psc.edu/machines/tcs/tcs.html

[12] http://archive.ncsa.uiuc.edu/SCD/Hardware/LinuxCluster/TechSummary/

[13] http://www.lanl.gov/orgs/pa/News/082300.html

parallel machines in this range by 2003. To scale NAMD to such large machines, we will develop and implement new parallel algorithms. Fine-grained decomposition techniques will be used to allow each time step to be completed in less than a millisecond for simulations involving over 100,000 atoms. We will experiment with variations of our hybrid spatial-and-force decomposition strategy that lead to smaller computational pieces. This fine grained decomposition will be supported by a new low-overhead runtime system, which can schedule each computational object with 1–2 microseconds overhead. Total communication on such large machines is typically limited because of physical considerations. We will develop new load balancers that place communicating objects in such a way as to minimize the average number of hops traveled by messages, while simultaneously minimizing load imbalance and communication volume.

Processor-in-memory architectures. During years 3–5 of the proposed funding period, machines with processors-in-memory chips are expected to become available. Such machines will provide a much higher aggregate bandwidth between processor and memory modules, by integrating multiple processors and memory within one chip. We will harness the high performance potential of such machines by developing a version of NAMD which supports a low memory-to-processor ratio (by eliminating replicated molecule data, for example). New algorithms that can deal with on-demand movement of data among memory modules will be implemented, to support prefetching of data. One of the potential machines in this category will be a "Teraflop Board" consisting of multiple multiprocessor chips designed for the original IBM Blue Gene machine.[14]

Accommodating large biomolecular aggregates. Larger simulations generally scale better on multi-processor machines than smaller simulations simply because there is more computation to be distributed. However, the memory usage of the master node in NAMD 2 is proportional to the number of atoms in the simulation, and does not decrease as additional processors are added. Because of this, systems larger than 100,000 atoms cannot run on a Cray T3E with only 128 MB of memory per node and 300,000 atoms can require 512 MB on the master node to perform load balancing. To address this problem, a parallel load balancing algorithm will be implemented that will not collect all load data on the master node. Additionally, the replicated molecular structure will be compressed by a general method which locates and eliminates repeated patterns such as water molecules, lipids, and amino acids.

6 Parallel Architecture Design

> *It is immensely ingenious, immensely complicated, and extremely effective, but somehow at the same time crude, wasteful, and inelegant, and one feels that there must be a better way of doing things.*
>
> *—C. Strachey (1962, concerning the IBM Stretch computer)*

When developing complex software, the first forays into a new application domain or approach to software design often lead to a collection of code which, although correct, is more indicative of the learning process of its creator than the final understanding which was gained. Successive implementations do not avoid this fate, but merely encounter it in the pursuit of new and more complex functionality.

[14] `http://www.research.ibm.com/bluegene/comsci.html`

NAMD 2 demonstrated that object oriented design and cooperative multithreading make it possible to write and maintain a complex message-driven program. However, a cleansing of the source code with the implementation of NAMD 3 will prevent its degeneration into the prevalent software architecture recently described as "a big ball of mud."[15] Insights to be applied in the design of NAMD 3 are summarized below.

Threadless control structure. NAMD 2 uses cooperative user-space threads to provide individual control loops for each fixed-size spatial domain (or *patch*) and for global aspects of the algorithm such as pressure control. These threads are scheduled by Charm++/Converse in the same queue as other tasks. In order to improve portability and allow for simplified debugging, control loops will be implemented using a simple state machine and preprocessor. Using object member variables in place of a stack, and providing additional control logic to handle dependencies, will allow for the clear expression of algorithms in this restrictive environment.

Exploiting multiprocessor nodes. NAMD 3 will run on clusters with multiprocessor nodes either with threads or as independent processes. Tasks will be permanently assigned to a given processor based on periodic load balancing. The alternative method of using a single work queue for all processors on a node would require a complex protocol to ensure that all processors completed at the same time and would also depend heavily on expensive synchronization and mutual exclusion functions. The primary concern for multiprocessor nodes will be optimizing data layout and dependencies to limit inter-node communication, a development which is not dependent on shared memory protocols.

Patterns for parallel operations. The parallel efficiency of NAMD 2 is due to a limited number of parallel operations, some of which are necessarily complex. The remainder of the code, however, only affects startup but is implemented via a variety of methods which were selected based on immediate convenience. The implementation and maintenance of NAMD 3 will be greatly aided by selecting a small number of useful patterns and encapsulating them as templates for repeated use.

Distributed simulation components. The general molecular dynamics algorithms and other utility code employed in NAMD 3 will be separated from the parallel control structures in order to provide a basis of components which can be independently tested and reused in other applications such as VMD. The imposition of clearly defined interfaces between these components will also aid the development of test harnesses for NAMD.

References

[1] M. Banikazemi, R. K. Govindaraju, R. Blackmore, and D. K. Panda. MPI-LAPI: An efficient implementation of MPI for IBM RS/6000 SP systems. *IEEE Trans. Parallel and Distributed Systems*, 2001. In press.

[2] M. Berger and S. Bokhari. A partitioning strategy for nonuniform problems on multiprocessors. *IEEE Trans. Computers*, C-36:570–580, 1987.

[15] `http://www.devcentre.org/mud/mudmain.htm`

[3] B. R. Brooks, R. E. Bruccoleri, B. D. Olafson, D. J. States, S. Swaminathan, and M. Karplus. CHARMM: A program for macromolecular energy, minimization, and dynamics calculations. *J. Comp. Chem.*, 4:187–217, 1983.

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

[5] T. Darden, D. York, and L. Pedersen. Particle mesh Ewald. An N·log(N) method for Ewald sums in large systems. *J. Chem. Phys.*, 98:10089–10092, 1993.

[6] High Performance Computational Chemistry Group. NWChem, a computational chemistry package for parallel computers, version 4.0.1. http://www.emsl.pnl.gov:2080/docs/nwchem.

[7] 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. Comp. Phys.*, 151:283–312, 1999.

[8] L. V. Kalé and S. Krishnan. Charm++: Parallel programming with message-driven objects. In G. V. Wilson and P. Lu, editors, *Parallel Programming using C++*, pages 175–213. MIT Press, 1996.

[9] E. Lindahl, B. Hess, and D. van der Spoel. GROMACS 3.0: a package for molecular simulation and trajectory analysis. *J. Mol. Mod.*, 2001.
http://link.springer.de/link/service/journals/00894/contents/01/00045.

[10] M. Nelson, W. Humphrey, A. Gursoy, A. Dalke, L. Kalé, R. D. Skeel, and K. Schulten. NAMD – A parallel, object-oriented molecular dynamics program. *Int. J. Supercomp. Appl. High Perform. Comp.*, 10:251–268, 1996.

[11] S. J. Plimpton and B. A. Hendrickson. A new parallel method for molecular dynamics simulation of macromolecular systems. *J. Comp. Chem.*, 17(3):326–337, 1996.

[12] J. W. Ponder and F. M. Richards. An efficient Newton-like method for molecular mechanics energy minimization of large molecules. *J. Comp. Chem.*, 8:1016–1024, 1987.

[13] R. D. Skeel. Symplectic integration with floating-point arithmetic and other approximations. *Appl. Numer. Math.*, 29:3–18, 1999.

[14] J. Stone, J. Gullingsrud, P. Grayson, and K. Schulten. A system for interactive molecular dynamics simulation. In J. F. Hughes and C. H. Séquin, editors, *2001 ACM Symposium on Interactive 3D Graphics*, pages 191–194, New York, 2001. ACM SIGGRAPH.

[15] P. K. Weiner and P. A. Kollman. AMBER: Assisted model building with energy refinement. A general program for modeling molecules and their interactions. *J. Comp. Chem.*, 2(3):287–303, 1981.