NAMD: Biomolecular Simulation on Thousands of Processors

James C. Phillips* Gengbin Zheng† Sameer Kumar† Laxmikant V. Kalé†

Abstract

NAMD is a fully featured, production molecular dynamics program for high performance simulation of large biomolecular systems. We have previously, at SC2000, presented scaling results for simulations with cutoff electrostatics on up to 2048 processors of the ASCI Red machine, achieved with an object-based hybrid force and spatial decomposition scheme and an aggressive measurement-based predictive load balancing framework. We extend this work by demonstrating similar scaling on the much faster processors of the PSC Lemieux Alpha cluster, and for simulations employing efficient (order N log N) particle mesh Ewald full electrostatics. This unprecedented scalability in a biomolecular simulation code has been attained through latency tolerance, adaptation to multiprocessor nodes, and the direct use of the Quadrics Elan library in place of MPI by the Charm++/Converse parallel runtime system.

1 Introduction

NAMD is a parallel, object-oriented molecular dynamics program designed for high performance simulation of large biomolecular systems [6]. 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 4000 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]

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.

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

*Beckman Institute, University of Illinois at Urbana-Champaign.

†Department of Computer Science and Beckman Institute, University of Illinois at Urbana-Champaign.

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

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

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

0-7695-1524-X/02 $17.00 © 2002 IEEE

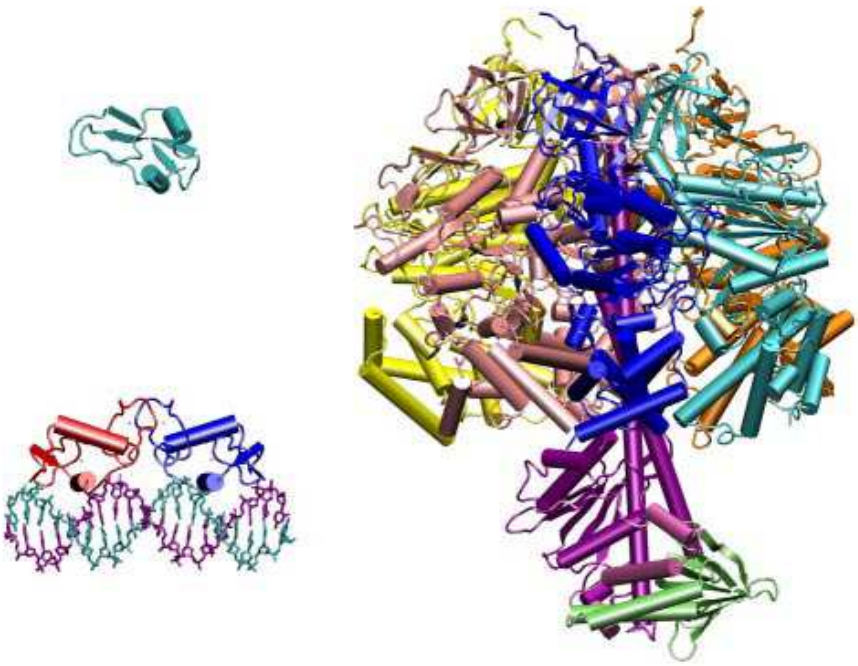

Figure 1: Simulations have increased exponentially in size, from BPTI (upper left, about 3K atoms), through the estrogen receptor (lower left, 36K atoms, 1996), to F_1-ATPase (right, 327K atoms, 2001). (Atom counts include solvent.)

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 [11].

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].

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 multi-threading and an object-oriented implementation in C++.

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 [3, 7]. 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 control the degree of parallelism. As described 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.

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, collects 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 strategies can also be developed by users and plugged in easily. In the following paragraphs, we will describe the load balancing strategies optimized for NAMD in detail.

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 92K atom ApoA1 benchmark, we only have 144 ($6 \times 6 \times 4$) 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 [10] 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. To further increase the number and reduce the granularity of these compute objects, they are split into subsets of interactions, each of roughly equal work.

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.

When running in parallel, some compute objects require data from patches not on the compute

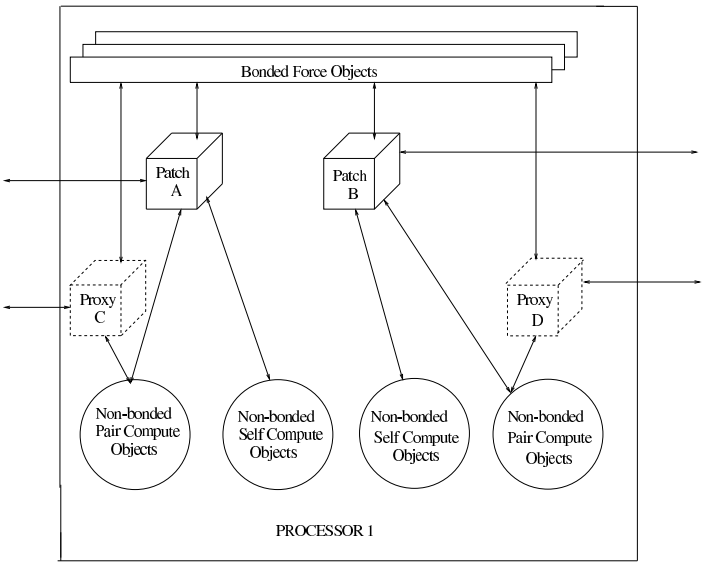

Figure 2: NAMD 2 hybrid force/spatial decomposition. Atoms are spatially decomposed into *patches*, which are represented on other nodes by *proxies*. Interactions between atoms are calculated by several classes of *compute objects*.

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.

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 [1], 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 time-steps (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.

The initial load balancer is aggressive, starting from the set of required proxies and assigning compute objects in order from larger to smaller, avoiding creating new proxies unless necessary. To assist this algorithm when the number of processors is larger than the number of patches, unoccupied processors are seeded with a proxy from a home patch on a close processor. In addition, when placing compute objects which only require a single patch (calculating interactions within that patch) the load balancer will prefer a processor with a proxy over the one with the actual home patch; reserving the home patch nodes for objects which need them more. After this initial balancing, only small refinements are made, attempting to transfer single compute objects off of overloaded processors without increasing communication. Two additional cycles of load balancing follow immediately in order to account for increased communication load, after which load balancing occurs periodically to maintain load balance.

3 Particle Mesh Ewald in NAMD

Particle mesh Ewald (PME) [5] full electrostatics calculations in NAMD have been parallelized in several stages, and this one feature has greatly affected the performance and scalability observed by users. As seen in Fig. 3, significant progress has been made. We initially incorporated the external DPME[4] 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, while elegant, was far from obvious, as numerous false starts were attempted along the way. In particular, the parallel implementation of FFTW[5] was found to be inappropriate for our purposes, due to an inefficient transpose operation designed to conserve memory. Instead, the serial components of the FFTW 3D FFT (2D and 1D FFTs) were used in combination with Charm++ messages to implement the data transpose. The reciprocal sum is currently parallelized only to the size of the PME grid, which is typically between 50 and 200. However, this is sufficient to allow NAMD simulations to scale efficiently, as the bottleneck has been significantly reduced.

The PME calculation involves five phases of calculation and four of communication, which the message-driven design of NAMD interleaves with other work, giving good performance even on high latency networks. The basic parallel structure is illustrated in Fig. 4. This overlap allows the FFT components of the PME operations and real-space direct force computations to execute on the same processor, in an interleaved fashion. When the number of processors is comparable to the number of PME processors (e.g. on 512 processors) this is a substantial advantage.

In the first and most expensive calculation, atomic charges are interpolated smoothly onto (typically) $4 \times 4 \times 4$ subsections of a three dimensional mesh; this is done on each processor for the atoms in the patches it possesses, as the process is strictly additive. Next, the grid is composited and decomposed along its first dimension to as many processors as possible. The spatial decomposition

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

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

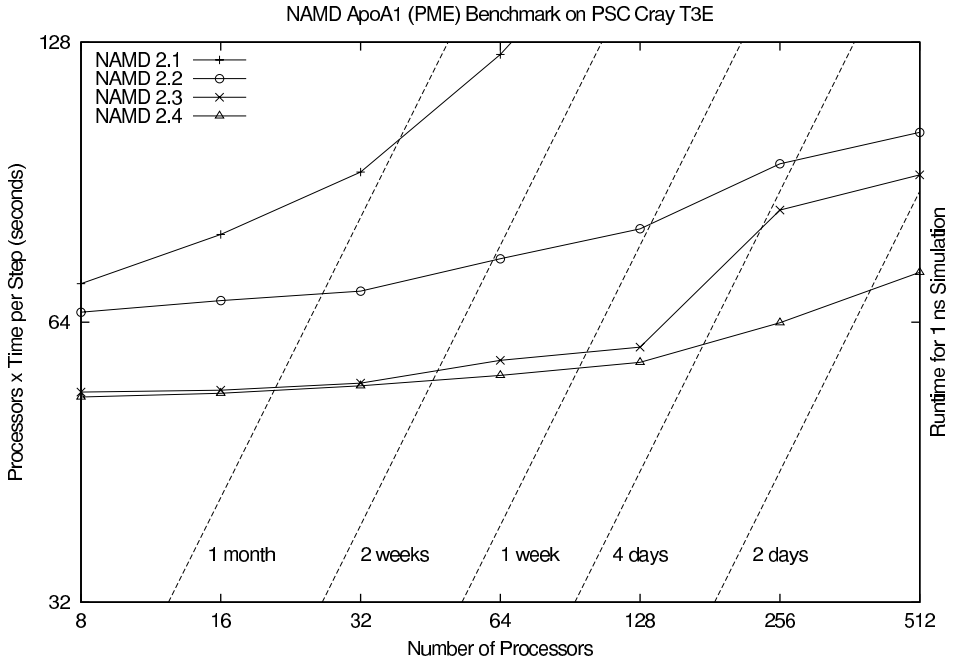

Figure 3: Total resources consumed per step for 92K atom benchmark with PME every four steps by NAMD versions 2.1–2.4 on varying numbers of processors of the PSC T3E. Perfect linear scaling is a horizontal line. Diagonal scale shows runtime per ns.

of the patches is used to avoid unnecessary messages or empty parts of the grid being transmitted. A local 2D FFT is performed by each processor on the second and third dimension of the grid, and the grid is then redistributed along its second dimension for a final 1D FFT on the first dimension. The transformed grid is then multiplied by the appropriate Ewald electrostatic kernel, and a backward FFT performed on the first dimension. The grid is redistributed back along its first dimension, and a backward 2D FFT performed, producing real-space potentials. The initial communication pattern is then reversed, sending the exact data required to extract atomic forces back to the processors with patches.

Improvements to the PME direct sum in NAMD 2.3 were obtained by eliminating expensive calls to `erfc()`.[6] 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.

Much of the improvement observed in NAMD 2.4 is due to the elimination of unnecessary messages and data, which could be excluded based on the overlap of patches and the PME grid. 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. Patches were similarly distributed, avoiding processors participating in PME when possible. This has increased the scalability of NAMD on the existing machines at NCSA, PSC,

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

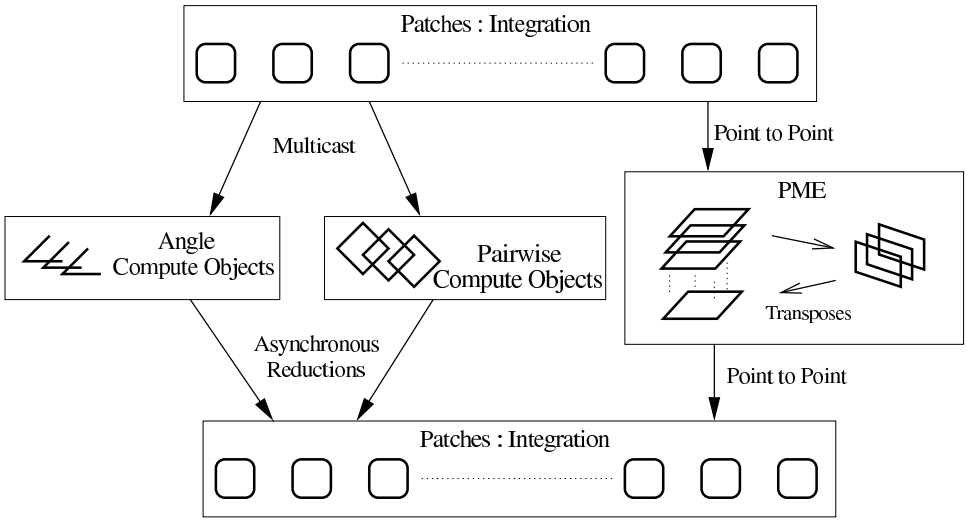

Figure 4: Parallel structure of PME in NAMD. When PME is required, typically every four steps, PME interleaves with other calculation.

and SDSC with 2, 4, and 8 processors per node. A more aggressive optimization would employ different sets of processors for each phase of the PME calculation on a sufficiently large machine. Figure 5 illustrates the portable scalability of NAMD 2.4 on a variety of recent platforms employed for production simulations by researchers at the Resource.

Continuing development for NAMD 2.5 as reported here has resulted in a more efficient interpolation table implementation and the elimination of unnecessary calculations from non-multiple-timestepping simulations, resulting in substantial improvements to serial performance. To improve locality of communication, patches and the PME grid planes with which they communicate have been aligned on the processors. In order to improve performance on Lemieux, various manipulations of the load balancer have been attempted, including removing all work from processors involved in the PME reciprocal sum, with limited success.

4 Optimization for PSC Lemieux

The Pittsburgh Supercomputing Center's Lemieux is a high performance Alpha cluster with 3000 processors and a peak performance of about 6 TFLOPS. Lemieux uses the Quadrics interconnection network, which surpasses other communication networks in speed and programmability [9]. Lemieux provides a default MPI library to access its interconnect. Our NAMD/Charm++ implementation on top of MPI uses MPI_Iprobe to get the size of the received message and then allocates a buffer for it. MPI_Iprobe turned out to be a very expensive call and affected the speedup of NAMD.

The network interface in Quadrics can also be accessed through the Elan communication library [12]. The Elan message passing library has a low latency of $5\mu s$ for small messages and a peak bandwidth of over 300 MB/s. The flexibility and programmability of Elan library motivated us to create a NAMD/Charm++ implementation directly on top of the Elan library. Our implementation has two types of messages, *small* and *large*, distinguished by different tags. Small messages are sent asynchronously along with the message envelope and are received in preallocated buffers. The maximum size of small messages and the number of preallocated buffers are a parameter to the

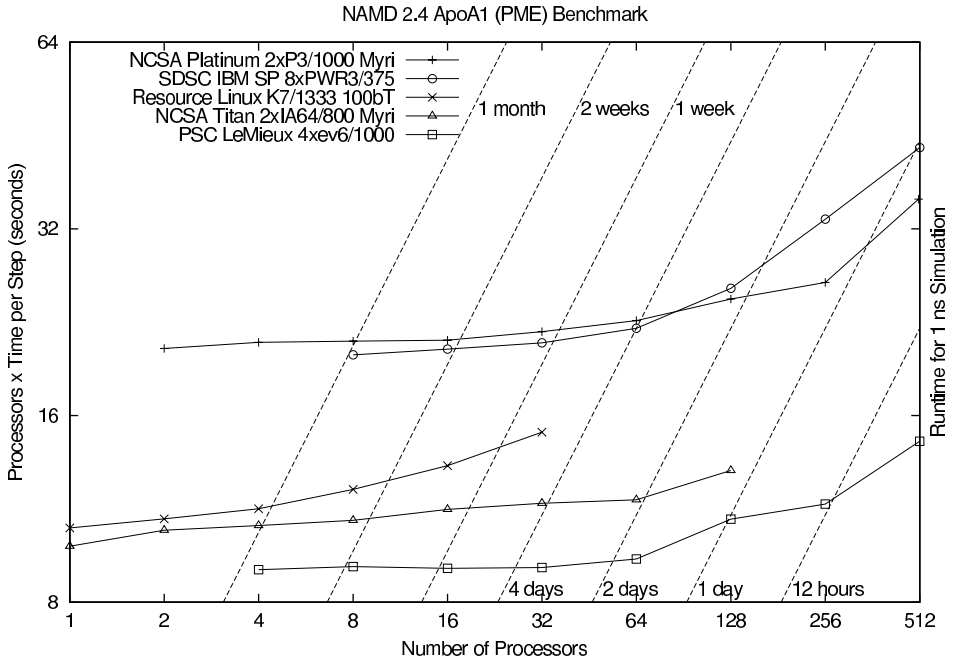

Figure 5: Total resources consumed per step for 92K atom ApoA1 PME MTS benchmark by NAMD 2.4 on varying numbers of processors for recent parallel platforms. Perfect linear scaling is a horizontal line. Diagonal scale shows runtime per ns.

program (we set the size of the buffers to about 5KB and number of buffers to the number of processors in the system). For large messages the sender first sends the message envelope and on receiving the message envelope the receiver allocates a buffer for the message and a DMA copy is performed from the sender's memory into the allocated buffer. The above framework is very easy to implement in Elan. A comparison of NAMD's performance on the MPI and Elan implementations is shown in Table 1.

The Elan network interface also allows user threads to be run on its processor. These threads could process the messages as soon as they are received and not disturb the main processors which are busy with computation. Thus multicast and reduction messages can be immediately sent to their destinations and other messages can be copied into local memory to be handled by the main processors when they become idle. This would greatly improve the efficiency of high level message passing systems like Charm++. This approach is currently being investigated.

We also implemented a message combining library that reduces the number of messages sent during the PME transposes from 192 or 144 for ATPase to 28. The library imposes a virtual mesh topology on the processors and each processor first combines and sends data to its column neighbors and then to its row neighbors. Thus each processor sends $2\sqrt{p}$ messages instead of p messages without the library. This helps in reducing the per message cost and is very useful for small messages. However notice that each byte is sent two times on the communication network. For ApoA1 the message size is around 600B and the library effectively reduces the step time. For ATPase the message size is around 900B which limits the gains of the library. For ATPase the library reduces the step time by about around 1ms for runs of over 2000 processors.

Processors		Time/step		Speedup		GFLOPS	
Total	Per Node	MPI	Elan	MPI	Elan	MPI	Elan
1	1	28.08 s	28.08 s	1	1	0.480	0.480
128	4	248.3 ms	234.6 ms	113	119	54	57
256	4	135.2 ms	121.9 ms	207	230	99	110
512	4	65.8 ms	63.8 ms	426	440	204	211
510	3	65.7 ms	63.0 ms	427	445	205	213
1024	4	41.9 ms	36.1 ms	670	778	322	373
1023	3	35.1 ms	33.9 ms	799	829	383	397
1536	4	35.4 ms	32.9 ms	792	854	380	410
1536	3	26.7 ms	24.7 ms	1050	1137	504	545
2048	4	31.8 ms	25.9 ms	883	1083	423	520
1800	3	25.8 ms	22.3 ms	1087	1261	521	605
2250	3	19.7 ms	18.4 ms	1425	1527	684	733
2400	4	32.4 ms	27.2 ms	866	1032	416	495
2800	4	32.3 ms	32.1 ms	869	873	417	419
3000	4	32.5 ms	28.8 ms	862	973	414	467

Table 1: NAMD performance on 327K atom ATPase benchmark system with and multiple timestepping with PME every four steps for Charm++ based on MPI and Elan.

5 Performance Analysis and Breakdown

One of the lessons from our previous parallel scaling effort was that we must keep the computation time of each individual object substantially lower than the per-step time we expect to achieve on the largest configuration. Since the heaviest computation happens in the pairwise force computation objects, we analyzed their grainsize, as shown in Fig. 6. This data was taken from 32 timesteps of a cutoff-only run on 2100 processors (although it will roughly be the same independent of the number of processors: the same set of objects are mapped to the available processors by Charm++ load balancing schemes). It shows that most objects' execution time is less than 800 microseconds, and the maximum is limited by 2.0 msecs or so. (The figure shows 905,312 execution instances of pairwise computation objects in all, over 32 steps. So, there are 28,291 pairwise compute objects per step. These objects account for about 845.53 seconds of execution time, or about 26.42 seconds per timestep. The average grainsize — the amount of computation per object — is about 934 microseconds).

Fig. 7 shows where the processors spent their time. As expected, the amount of time spent on communication, and related activities increases with number of processors. But the percentage increase is moderate, once the communication time has risen to a significant percentage at 128 processors. The idle time is due almost entirely due to load imbalances (since there are no blocking receives in Charm++), which are handled relatively efficiently by the measurement-based load balancing of Charm++.

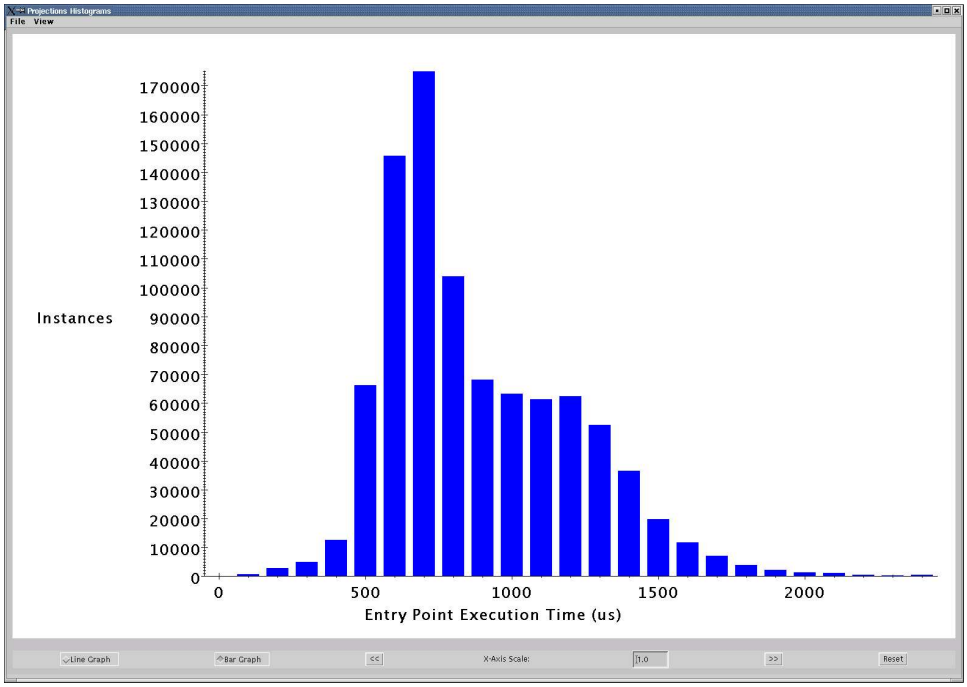

Figure 6: Grainsize of pairwise nonbonded computations on a 1536 processor ATPase cutoff run over 32 timesteps. Uniform grainsize aids load balancing and the interleaving of higher priority tasks. Figure generated by the projections utility of Charm++.

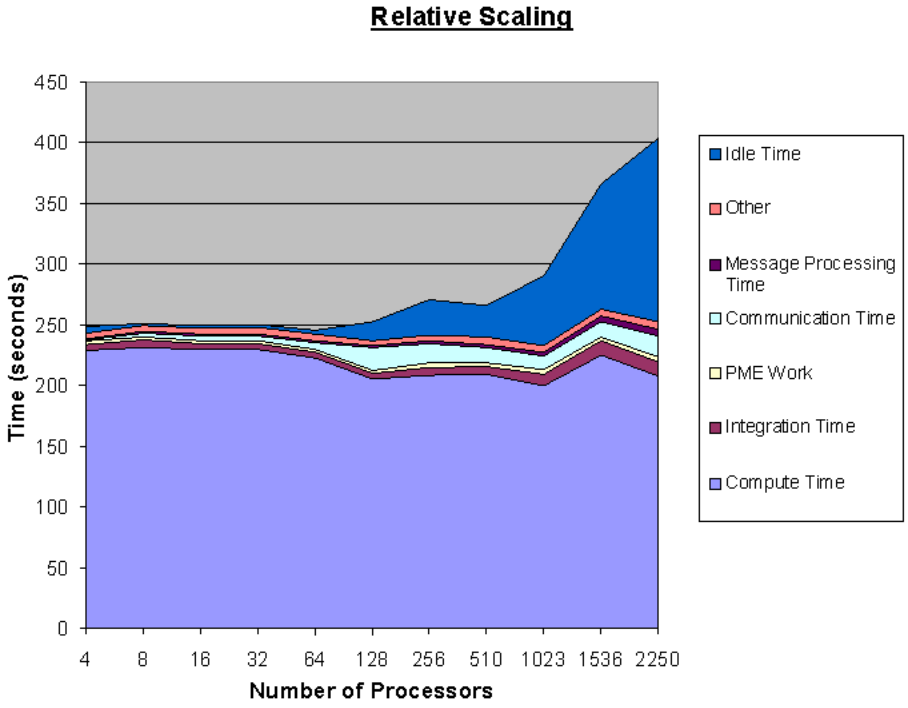

Figure 7: Time spent in various activities, during 8 timesteps of an ATPase PME MTS run as a function of processor count.

System	Atoms	GOP per step			GFLOP per step		
		Cut	PME	MTS	Cut	PME	MTS
ApoA1	92K	9.68	10.64	10.75	3.52	3.65	3.84
ATPase	327K	32.07	35.66	36.05	12.29	12.80	13.48

Table 2: Measured operation counts for NAMD on benchmark systems and simulation modes cutoff (Cut), PME every step (PME), and multiple timestepping with PME every four steps (MTS).

6 Benchmark Methodology

NAMD is a production simulation engine, and has been benchmarked against the standard community codes AMBER [13] and CHARMM [2]. The serial performance of NAMD is comparable to these established packages, while its scalability is much better. NAMD is an optimized and efficient implementation of the MD algorithm as applied to biomolecular systems.

In order to demonstrate the scalability of NAMD for the real problems of biomedical researchers, we have drawn benchmarks directly from simulations being conducted by our NIH-funded collaborators. The smaller *ApoA1* benchmark comprises 92K atoms of lipid, protein, and water, and models a high density lipoprotein particle found in the bloodstream. The larger *ATPase* benchmark is taken from simulations still in progress, consists of 327K atoms of protein and water, and models the F_1 subunit of ATP synthase, a component of the energy cycle in all life. Both systems comprise a solvated biomolecular aggregate and explicit water in a periodic cell. While we have favored larger simulations to demonstrate scalability, the simulations themselves are not gratuitously large, but were created by users of NAMD to be as small as possible but still scientifically valid. We have not scaled the benchmark simulations in any way, or created an artificially large simulation in order to demonstrate scalability.

The number of short-range interactions to be evaluated in a simulation, and hence its serial runtime, is proportional to the cube of the cutoff distance. The realistic selection of this value is vital to the validity of the benchmark, as excessive values inflate the ratio of computation to data. For all benchmarks here, short-range nonbonded interactions were cut off at 12 Å as specified by the CHARMM force field. The spatial decomposition employed in NAMD bases domain size on the cutoff distance and, therefore, NAMD will sometimes scale better with smaller cutoffs. Our 12 Å cutoff results in $6 \times 6 \times 4 = 144$ patches (domains) for ApoA1 and $11 \times 8 \times 8 = 704$ patches for ATPase; scaling is harder when more processors than patches are used.

The serial performance implications of the selection of the PME full electrostatics parameters are minor. Provided the dimensions of the charge grid have sufficient factors to allow $O(NlogN)$ scaling of the 3D FFT, the bulk of the work is proportional to the number of atoms and the number of points each charge is interpolated to. The default $4 \times 4 \times 4$ interpolation as used and the grid was set at a spacing of approximately 1 Å, $108 \times 108 \times 80$ for ApoA1 and $192 \times 144 \times 144$ for ATPase. In typical usage, full electrostatics interactions are calculated every four timesteps using a multiple timestepping integrator; such a simulation is of equivalent accuracy to one in which PME is evaluated every step. To observe the effect of PME on performance and scalability, we benchmark three variations: no PME or cutoff (Cut), PME every step (PME), and PME every four steps (MTS).

Operation counts for both benchmarks were measured using the *perfex* utility for monitoring the hardware performance counters on the SGI Origin 2000 at NCSA. Total and floating point operations were measured for all combinations of benchmark systems (ApoA1, ATPase) and sim-

ulation modes (Cut, PME, MTS) for runs of 20 and 40 steps running on a single processor (to eliminate any parallel overhead). The results of the 40 and 20 step simulations were subtracted to eliminate startup calculations and yield an estimate of the marginal operations per step for a continuing simulation. The results of this calculation are presented in Table 2 and used as the basis for operation counts in parallel runs.

Based on serial runtimes, NAMD executes around 0.5 floating point operations and 1.3 total operations per clock cycle on the 1 GHz processors of Lemieux. The ratio of integer to floating point operations observed reflects the efficiency of NAMD in using efficient algorithms to avoid unnecessary floating point operations whenever possible. For example, a recent optimization manipulates an IEEE float as an integer in order to index into the short-range electrostatics interpolation table.

Measured by operation count, PME is only slightly more expensive than cutoff calculations. This is due to the $O(N \log N)$ complexity of the long-range calculation and the efficient incorporation of the short-range correction into the electrostatic interaction. Operation counts are nearly proportional to the number of atoms in the simulation, demonstrating the near-linear scaling of the PME algorithm. The additional operations in MTS (despite fewer PME calculations!) are due to the need to separate the short-range correction result rather than including it in the short-range results. Note, however, that the actual runtime of MTS is less than that of PME, since the short-range correction is calculated very efficiently.

For scaling benchmark runs, execution time is measured over the final 2000 steps of a 2500 step simulation. Initial load balancing occurs during the first 500 steps, broken down as 100 steps ignored (to eliminate any lingering startup effects), 100 steps of measurement and a complete reassignment, 100 steps of measurement and a first refinement, another 100 steps of measurement and a second refinement, and finally another 100 ignored steps. In a continuing simulation, load balancing would occur every 4000 steps (i.e., at steps 4400, 8400, etc.), preceded immediately by 100 steps of measurement.

In order to demonstrate that the reported scaling and performance is sustainable in a production simulation, we have run the ApoA1 MTS benchmark for 100,000 steps on 512 processors of Lemieux, recording the average time per step for each 100 step interval. This number of processors was chosen to ensure that performance was heavily influenced by the load balancer rather than some other bottleneck in the calculation. As seen in Figure 8, time spent load balancing is hardly perceptible, while the benefits of periodic refinement over the course of the run are substantial. The average performance over all 100,000 steps including initial load balancing is 23.3 ms/step, while the average for steps 500-2500 as employed in speedup calculations is 26.3 ms/step, some 10% slower than the performance experienced by an actual user. The value reported for the matching 2500 step run in the speedup table is 23.9 ms/step, which is again higher than the overall average seen here. We can therefore confidently state that the results reported below are conservative and indicative of the true performance observed during a production run.

7 Scalability Results

Table 3 shows the performance of NAMD on the smaller ApoA1 benchmark system. While reasonable efficiency is only obtained to 512 processors, by using larger numbers of processors the time per step can be reduced to 11.2 ms cutoff and 12.6 ms PME (MTS). The ApoA1 data also reveals superior scaling when only three of the four processors per node available on Lemieux are used.

Table 4 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

Processors		Time/step			Speedup			GFLOPS		
Total	Per Node	Cut	PME	MTS	Cut	PME	MTS	Cut	PME	MTS
1	1	7.08 s	8.24 s	7.86 s	1	1	1	0.497	0.443	0.489
128	4	62.5 ms	80.0 ms	71.5 ms	113	103	109	56	45	53
256	4	37.4 ms	43.7 ms	40.3 ms	189	188	194	94	83	95
512	4	21.0 ms	26.6 ms	23.9 ms	336	309	329	167	137	161
510	3	20.5 ms	24.9 ms	23.3 ms	344	331	337	171	146	164
1024	4	18.2 ms	24.4 ms	17.6 ms	389	338	446	193	149	218
1023	3	13.8 ms	15.2 ms	14.0 ms	512	540	559	254	239	273
1536	4	16.6 ms	22.6 ms	17.1 ms	427	364	459	212	161	224
1536	3	11.2 ms	15.0 ms	12.6 ms	629	549	624	312	243	305
2250	3	11.2 ms	15.0 ms	12.8 ms	629	549	613	313	243	299

Table 3: NAMD performance on 92K atom ApoA1 benchmark system with simulation modes cutoff (Cut), PME every step (PME), and multiple timestepping with PME every four steps (MTS) for Charm++ based on Elan.

Processors		Time/step			Speedup			GFLOPS		
Total	Per Node	Cut	PME	MTS	Cut	PME	MTS	Cut	PME	MTS
1	1	24.89 s	29.49 s	28.08 s	1	1	1	0.494	0.434	0.480
128	4	207.4 ms	249.3 ms	234.6 ms	119	118	119	59	51	57
256	4	105.5 ms	135.5 ms	121.9 ms	236	217	230	116	94	110
512	4	55.4 ms	72.9 ms	63.8 ms	448	404	440	221	175	211
510	3	54.8 ms	69.5 ms	63.0 ms	454	424	445	224	184	213
1024	4	33.4 ms	45.1 ms	36.1 ms	745	653	778	368	283	373
1023	3	29.8 ms	38.7 ms	33.9 ms	835	762	829	412	331	397
1536	4	25.7 ms	44.7 ms	32.9 ms	968	660	854	477	286	410
1536	3	21.2 ms	28.2 ms	24.7 ms	1175	1047	1137	580	454	545
2048	4	25.8 ms	46.7 ms	25.9 ms	963	631	1083	475	274	520
1800	3	18.6 ms	25.8 ms	22.3 ms	1340	1141	1261	661	495	605
2250	3	15.6 ms	23.5 ms	18.4 ms	1599	1256	1527	789	545	733
2400	4	22.6 ms	44.6 ms	27.2 ms	1099	661	1032	542	286	495
2800	4	22.1 ms	43.6 ms	32.1 ms	1127	676	873	556	293	419
3000	4	22.6 ms	39.6 ms	28.8 ms	1102	743	973	544	322	467

Table 4: NAMD performance on 327K atom ATPase benchmark system with simulation modes cutoff (Cut), PME every step (PME), and multiple timestepping with PME every four steps (MTS) for Charm++ based on Elan. Peak performance is attained on 2250 processors, three per node.

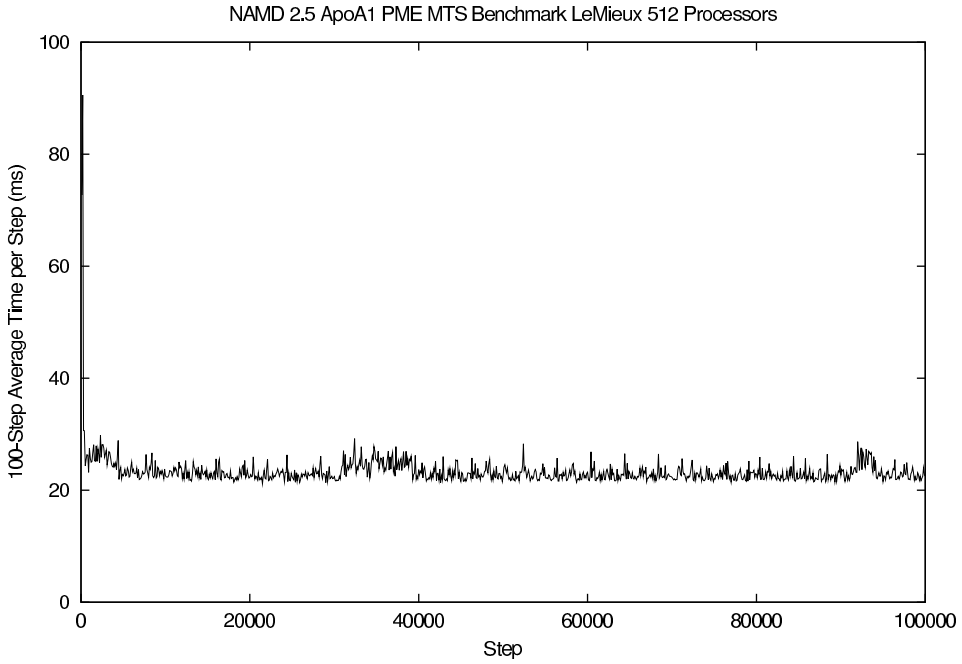

Figure 8: Performance variation over a longer simulation of ApoA1 MTS on 512 processors of Lemieux. Periodic load balancing maintains and improves performance over time. Overall average performance is 23.3 ms/step, while performance measured for steps 500-2500 of this run is 26.3 ms/step and performance reported in speedup tables for a 2500 step run is 23.9 ms/step.

performance is most greatly desired by researchers. Maximum performance is achieved employing 2250 processors, three processors per node for all of the 750 nodes of Lemieux. For the cutoff simulation, a speedup of 1599 at 789 GFLOPS was attained. For the more challenging PME (MTS) simulation, a speedup of 1527 at 733 GFLOPS was attained. This 68% efficiency for a full electrostatics biomolecular simulation on over 2000 processors indicates that the scalability problems imposed by PME have been overcome.

8 Remaining Challenges

As can be seen above, we have achieved quite satisfactory performance levels when running on up to 2250 processors. The major problem remaining appears to be simply the inability to use the 4th processor on each node for useful computation effectively. As the speedup data shows, we consistently get smaller speedups (and slowdowns) when we try to use all 4 processors on a node.

Even more fundamentally, our analysis shows that the problems are due to "stretching" of various operations (typically inside send or receive operations inside the communication layers). The message driven nature of NAMD and Charm++ allows us to effectively deal with small variations of this nature. For example, Fig. 9 shows the "timeline" view using the *projections* performance tool, for a run on 1536 processors (using 512 nodes). As can be seen, processors 900 and 933 had "stretched" events, even on this run that used 3 out of 4 processors on each node. However, other processes reordered the computations they were doing (automatically, as a result of data-driven execution) so that they were not held up for data, and the delayed processors "caught

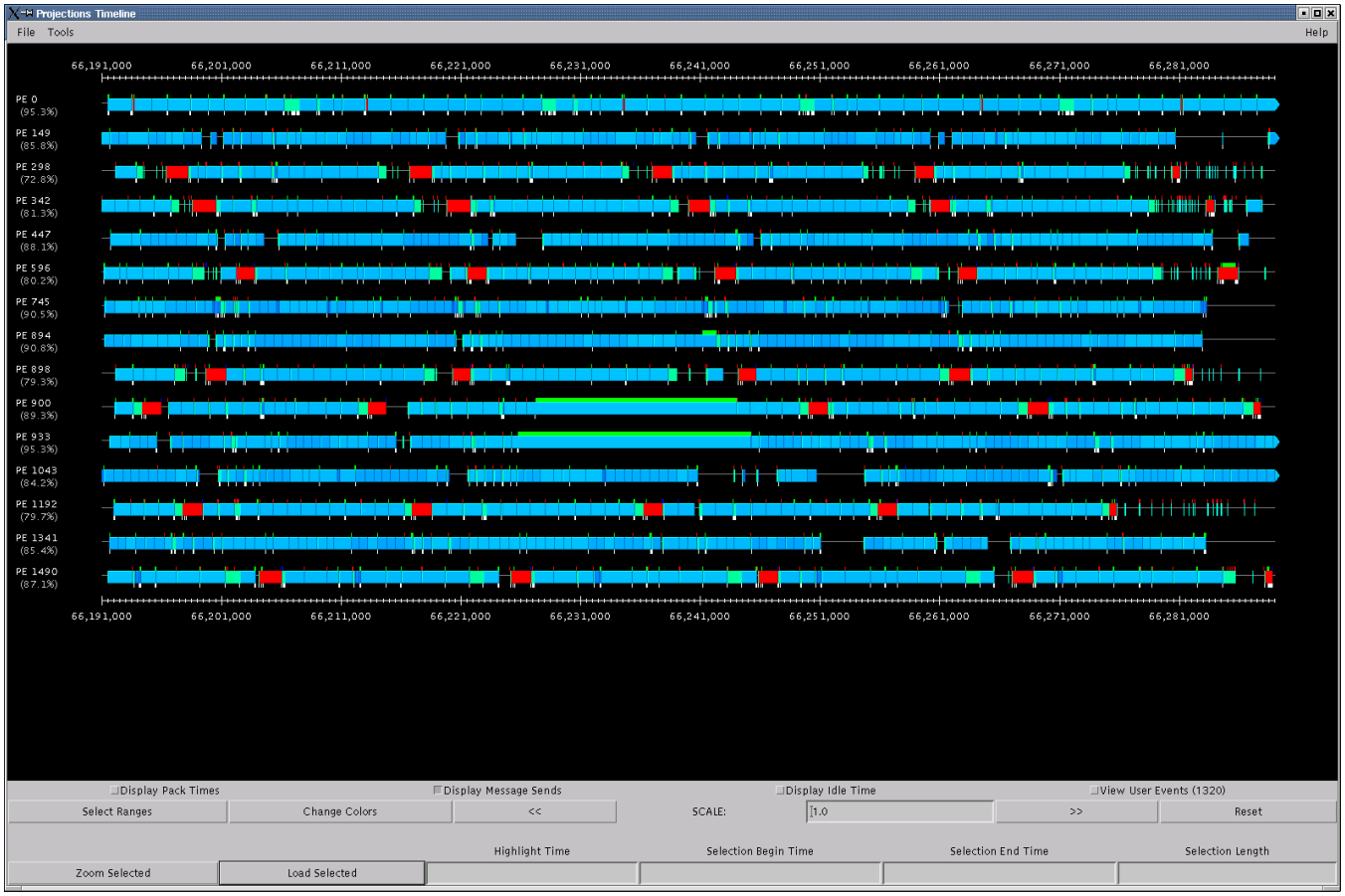

Figure 9: Projections timeline view illustrating harmful stretching and message-driven adaptation on 1536 processor cutoff run using three processors per node.

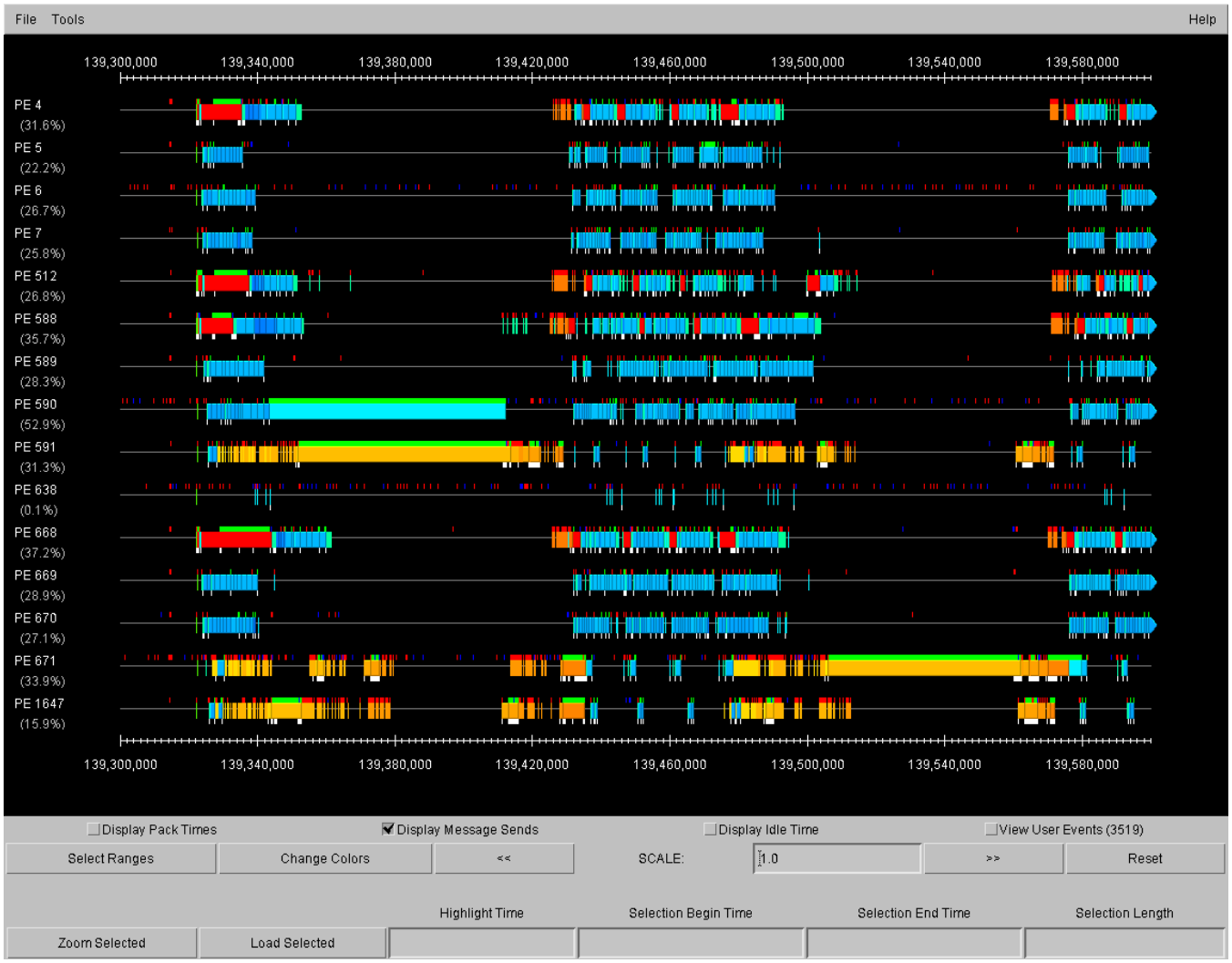

Figure 10: Projections timeline view illustrating catastrophic stretching on 3000 processor PME MTS run using four processors per node.

up" without causing a serious dent in performance. On the other hand, With 4 processors per node, such stretching events happen more often, making hiding them via data-driven adaptation impossible. Further, with PME, which imposes a global synchrony across all patches, processors have less leeway to adjust for the misbehavior. The compound result of these two factors is the substantial degradation in performance seen in Fig. 10. This is the reason for the better speedups for cutoff-based runs on 2800 and 3000 processors in Table 4.

We are working with PSC and HP staff to address these problems. We are especially hopeful that a different implementation of our low level communication primitives will overcome this problem, and allow our application to utilize the entire set of processors of the machine effectively.

Parallelizing the MD algorithm to thousands of processors reduces individual timesteps to the range of 5–25 ms. The difficulty of this task is compounded by the speed of modern processors, e.g., ASCI Red ran the ApoA1 benchmark without PME at 57 s per step on one processor while a 1 GHz Alpha of Lemieux runs with PME at 7.86 s per step. The scalability results reported here demonstrate the strengths and limitations of both the software, NAMD and Charm++, and the hardware, Lemieux.

Acknowledgements

NAMD was developed as a part of biophysics research at the Theoretical Biophysics Group (Beckman Institute, University of Illinois), which operates as an NIH Resource for Macromolecular Modeling and Bioinformatics. This resource is led by principle investigators Professors Klaus Schulten (Director), Robert Skeel, Laxmikant Kalé and Todd Martinez. We are thankful for their support, encouragement, and cooperation. Prof. Skeel and coworkers have designed the specific numerical algorithms used in NAMD. We are grateful for the funding to the resource provided by the National Institutes of Health (NIH PHS 5 P41 RR05969-04).

NAMD itself is a collaborative effort. NAMD 1 was implemented by a team including: Robert Brunner, Andrew Dalke, Attila Gursoy, Bill Humphrey and Mark Nelson. NAMD 2, the current version, was implemented and is being enhanced by by a team consisting of: Milind Bhandarkar, Robert Brunner, Paul Grayson, Justin Gullingsrud, Attila Gursoy, David Hardy, Neal Krawetz, Jim Phillips, Ari Shinozaki, Krishnan Varadarajan, Gengbin Zheng, and Fangqiang Zhu.

This research has benefited directly from the Charm++ framework at the Parallel Programming Laboratory (http://charm.cs.uiuc.edu), and especially its load balancing framework and strategies, the work on the performance tracing and visualization tool *projections*), and its recent extensions for using chip-level performance counters. For help with the results in this papers, as well as for relevant work on Charm++, we thank Orion Lawlor, Ramkumar Vadali, Joshua Unger, Chee-Wai Lee, and Sindhura Bandhakavi.

Charm++ framework is also being used and supported by the Center for Simulation of Advanced Rockets (CSAR or simply the Rocket Center) at the University of Illinois at Urbana-Champaign, funded by the Department of Energy (via subcontract B341494 from Univ. of California,), and the NSF NGS grant (NSF EIA 0103645) for developing this programming system for even larger parallel machines extending into PetaFLOPS level performance.

The parallel runs were carried out primarily at the Pittsburgh Supercomputing Center (PSC) and also at the National Center for Supercomputing Applications (NCSA). We are thankful to these organizations and their staff for their continued assistance and for the early access and computer time we were provided for this work. In particular we would like to thank David O'Neal, Sergiu Sanielevici, John Kochmar and Chad Vizino from PSC and Richard Foster (Hewlett Packard)

for helping us make the runs at PSC Lemieux and providing us with technical support. Computer time at these centers was provided by the National Resource Allocations Committee (NRAC MCA93S028).

References

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

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

[3] R. K. Brunner and L. V. Kalé. Adapting to load on workstation clusters. In *The Seventh Symposium on the Frontiers of Massively Parallel Computation*, pages 106–112. IEEE Computer Society Press, February 1999.

[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 sumsin large systems. *J. Chem. Phys.*, 98:10089–10092, 1993.

[6] 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.

[7] L. V. Kale, M. Bhandarkar, and R. Brunner. Run-time Support for Adaptive Load Balancing. In J. Rolim, editor, *Lecture Notes in Computer Science, Proceedings of 4th Workshop on Runtime Systems for Parallel Programming (RTSPP) Cancun - Mexico*, volume 1800, pages 1152–1159, March 2000.

[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] F. Petrini, W. chun Feng, A. Hoisie, S. Coll, and E. Frachtenberg. The quadrics network: High-performance clustering technology. *IEEE Micro*, 22(1):46– 57, 2002.

[10] 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.

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

[12] Elan programming manual. http://www.lanl.gov/radiant/website/pubs/quadrics/qsnet.pdf.

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