NAMD: A Portable and Highly Scalable Program for Biomolecular Simulations

Abhinav Bhatele[1], Sameer Kumar[3], Chao Mei[1], James C. Phillips[2], Gengbin Zheng[1], Laxmikant V. Kalé[1]

[1] Department of Computer Science, University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA
{bhatele2, chaomei2, gzheng, kale}@.uiuc.edu

[2] Beckman Institute, University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA
jim@ks.uiuc.edu

[3]IBM T. J. Watson Research Center
Yorktown Heights, NY 10598, USA
sameerk@us.ibm.com

Abstract

NAMD is a portable parallel application for biomolecular simulations. NAMD pioneered the use of hybrid spatial and force decomposition, a technique used by most scalable programs for biomolecular simulations, including Blue Matter and Desmond which were described at Supercomputing 2006. This paper describes parallel techniques and optimizations developed to enhance NAMD's scalability, to exploit recent large parallel machines. NAMD is developed using Charm++ and benefits from its adaptive communication-computation overlap and dynamic load balancing, as demonstrated in this paper. We describe some recent optimizations including: pencil decomposition of the Particle Mesh Ewald method, reduction of memory footprint, and topology sensitive load balancing. Unlike most other MD programs, NAMD not only runs on a wide variety of platforms ranging from commodity clusters to supercomputers, but also scales to thousands of processors. We present results for up to 32,000 processors on machines including IBM's Blue Gene/L, Cray's XT3, and TACC's LoneStar cluster.

1 Introduction

Molecular Dynamics (MD) simulations of biomolecular systems constitute an important technique for our understanding of biological systems, exploring the relationship between structure and function of biomolecules, and rational drug design. Yet such simulations are highly challenging to parallelize because of the relatively small number of atoms involved, and extremely large time-scales. Due to the high frequencies of bond vibrations, a time-step is typically about 1 femtosecond (fs). Biologically important phenomena require a time scale of at least hundreds of nanoseconds (ns), or even a few microseconds (us). In contrast to billion-atom simulations needed in material science,

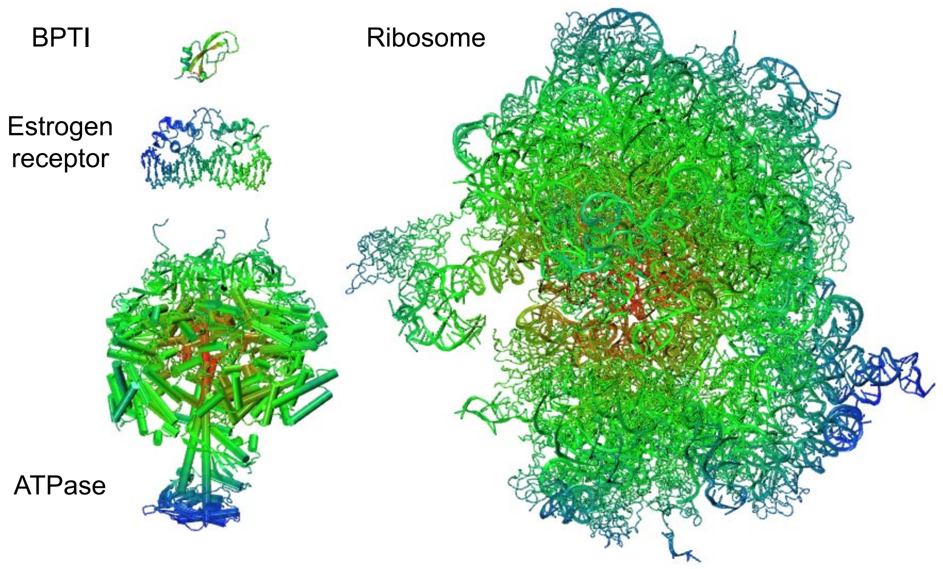


Figure 1: The size of biomolecular system that can be studied through all-atom simulation has increased exponentially in size, from BPTI (bovine pancreatic trypsin inhibitor, upper left, about 3K atoms), through the estrogen receptor (middle left, 36K atoms, 1996) and F_1-ATPase (lower left, 327K atoms, 2001), to the ribosome (right, 2.8M atoms, 2006). Atom counts include solvent, not shown for better visualization.

biological systems require simulations involving only tens of thousands of atoms to a few million atoms, as illustrated in Fig. 1.

Application scientists (biophysicists) would like to run their simulations on any of the available machines at the national centers, and would like to be able to checkpoint simulations on one machine and then continue on another machine. Scientists at different laboratories/organizations typically have access to different types of machines. An MD program that performs well across a range of architectures is therefore desirable.

A new challenge is added by the impending arrival of petascale machines: NSF recently announced plans to deploy a machine with sustained petascale performance by 2011 and provided a biomolecular simulation benchmark with 100M atoms as one of the applications that must run well on such a machine. When combined with lower per-core memory (in part due to high cost of memory) on machines such as Blue Gene/L, this poses the challenge of fitting within available memory.

This paper describes parallelization techniques that have led to high scaling and performance on a wide variety (size) of benchmarks. One common theme is that different machines, different number of processors, and different molecular systems may require a different choice or variation of algorithm. A parallel design that is flexible and allows the runtime system to choose between such alternatives, and a runtime system capable of making intelligent choices adaptively is required to attain high performance over such a wide terrain of parameter space. Also, we show that the use of measurement based dynamic load balancing is useful in simultaneously minimizing load imbalance, reducing communication overhead and reducing communication contention.

We also showcase our performance on different machines including Cray XT3 (up to 4000 processors), Blue Gene/L (up to 32,768 processors), TACC Lonestar system (up to 1,024 processors) and SDSC DataStar cluster (up to 1,024 processors) on molecular systems ranging in size from 5.5K atoms to 2.8M atoms, although not on all combinations of machine-size and number-of-atoms. We also compare with other MD programs which have been developed recently such as Blue Matter and Desmond.

We first describe the basic parallelization methodology adopted by NAMD. We then describe a series of optimizations, alternative algorithms, and performance tradeoffs that were developed to enhance the scalability of NAMD. For many of these techniques, we provide detailed analysis that may be of use to MD developers as well as other parallel application developers. We then showcase the performance of NAMD for different architectures and compare it with other MD programs. We then summarize the contributions in the paper and also talk about the simulations done using NAMD (see Appendix).

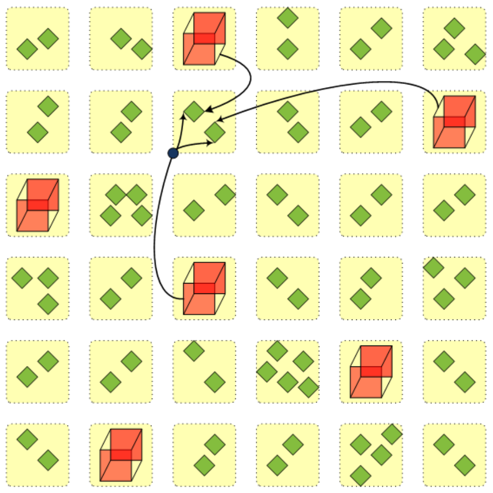
2 NAMD Parallelization Strategy

Classical molecular dynamics requires computation of two distinct categories of forces: (1) Forces due to bonds (2) Non-bonded forces. The non-bonded forces include electrostatic and Van der Waal's forces. A naive evaluation of these forces will take O(N^2) time. However, by using a cut-off radius r_c and separating the calculation of short-range and long-range forces, one can reduce the asymptotic operation count to O($N \log N$). Forces between atoms within r_c are calculated explicitly (this is an O(N) component, although with a large proportionality constant). For long-range calculation, the Particle-Mesh Ewald algorithm is used, which transfers the electric charge of each atom to electric potential on a grid and uses a 3D FFT to calculate the influence of all atoms on each atom. Although the overall asymptotic complexity is O($N \log N$), the FFT component is often smaller, and the computation time is dominated by the O(N) computation.

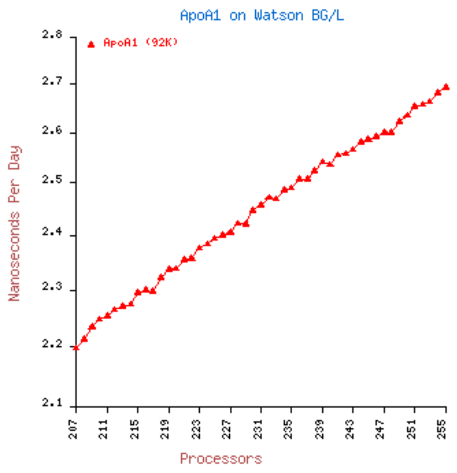
Prior to [5], parallel biomolecular simulation programs used either atom decomposition, spatial decomposition, or force decomposition (for a good survey, see Plimpton et. al [8]). NAMD was one of the first programs to use a hybrid of spatial and force decomposition that combines the advantages of both. More recently, methods used in Blue Matter [3, 4], the neutral territory and midpoint territory methods of Desmond [1], and those proposed by Snir [9] use variations of such a hybrid strategy.

NAMD decomposes atoms into boxes called "cells" (see Figure 2(a)). The size of each cell d_{min} along every dimension, is related to r_c. In the simplest case (called "1-Away" decomposition), d_{min}= $r_c + margin$, where the margin is a small constant. This ensures that atoms that are within cutoff radius of each other stay within the neighboring boxes over a few (e.g. 20) time steps. The molecular system being simulated is in a simulation box of dimensions $B_x \times B_y \times B_z$ typically with periodic boundary conditions. So, the size of a cell along each dimension d_x is the smallest d, such that B_x/m is greater than d_{min}, for some integer m. So, if $r_c = 12$ and $margin = 4$, $d_{min} = 16$, the cells should be of size $16 \times 16 \times 16$. However, if the simulation box is $108.86 \times 108.86 \times 77.76$ Å in the case of ApoLipoprotein-A1 simulation, this is impossible. Since the simulation box along X dimension is 108.86 Å, one must pick $108.86/6 = 18.15$ Å as the size along X axis for the cell. (And the size of a cell will be $18.15 \times 18.15 \times 19.44$). This is the spatial decomposition component of the hybrid strategy.

For every pair of interacting cells, (in our 1-Away decomposition, that corresponds to every pair of touching cells), we create an object (called a "compute object" or just "compute") whose



(a)



(b)

Figure 2: (a) Placement of cells and computes on a 2D mesh of processors, (b) Performance of NAMD on consecutive processor numbers

responsibility is to compute the pairwise forces between the two cells. This is the force decomposition component. Since each cell has 26 neighboring cells, one gets $14 \times C$ compute objects ($26/2+1 = 14$), where C is the number of cells. These compute objects are assigned to processors by a dynamic load balancer (see Section 2.2). This gives our program the ability to run on a range of differing number of processors, without changing the decomposition.

When the ratio of atoms to processors is smaller, we decompose the cells further. In general, along each axis X, the size of a cell can be d_{min}/k where k is typically 1 or 2 (and rarely 3). Since the cells that are "2-away" from each other must interact if k is 2 along a dimension, this decomposition is called 2-away-X, 2-away-XY or 2-awayXYZ etc. depending on which dimension uses $k = 2$.

Neither the number of cells nor the number of compute objects need to be equal to the exact number of processors. Typically, the number of cells is smaller than the number of processors, by an order of magnitude, which still generates adequate parallelism (because of the separation of "compute" objects) to allow the load balancer to optimize communication, and distribute work evenly. As a result, NAMD is able to exploit any number of available processors. Figure 2(b) shows the performance of the simulation of ApoA1 system (details in section 4) on varying numbers of processors in the range 207-255. In contrast, schemes that decompose particles into P boxes, where P is the total number of processors may limit the number of processors they can use for a particular simulation: they may require P to be a power of two or be a product of three numbers with a reasonable aspect ratio.

A careful load balancer assigns compute objects to processors so as to minimize the number of messages exchanged, in addition to minimizing load imbalance. As a result, NAMD is typically able to use less than 20 messages per processor (10 during multicasting of coordinates to compute objects, and 10 to return forces). Table 1 shows the number of messages (in the non-PME portion

Processors	**512**	**1024**	**1024**	**2048**	**4096**	**8192**	**8192**	**16384**	**20480**
Decomposition	X	X	XY	XY	XY	XY	XYZ	XYZ	XYZ
No. of Messages	4797	7577	13370	22458	29591	35580	79285	104469	110515
Message Size (bytes)	9850	9850	4761	4761	4761	4761	2303	2303	2303
Comm. Volume (MB)	47.3	74.6	63.7	107	141	169	182	241	254
Atoms per patch	296	296	136	136	136	136	60	60	60
Time step (ms)	18.6	11.6	10.3	6.9	4.7	3.2	3.22	2.33	2.24

Table 1: Communication Statistics on IBM BG/L

of the computation) as a function of number of processors for the ApoA1 benchmark simulation. The number of cells and the decomposition used is also shown. The load balancer and the use of spanning trees (Section 3.4) ensures that the variation in actual number of messages sent/received by different processors is small, and they are all close to the average number. The size of each message is typically larger than that used by Desmond and Blue Matter. Since many modern parallel machines use RDMA capabilities, which emphasize per message cost, and hide the per byte cost (by off-loading communication to co-processors), we believe this to be a better tradeoff.

We now describe a few features of NAMD and analyze how they are helpful in scaling performance to a large number of processors.

2.1 Adaptive Overlap of Communication and Computation

NAMD uses a message-driven runtime system to ensure that multiple modules can be composed concurrently without losing efficiency. In particular, idle times in one module can be exploited by useful computations in another. Furthermore, NAMD uses asynchronous reductions, whenever possible (such as in the calculation of energies). As a result, the program is able to continue without sharp reductions in utilization around barriers. For example, Figure 6 shows a time profile of a simulation of ApoA1 (see Table 5) on 1024 processors on Blue Gene/L (This figure was obtained by using the Performance analysis tool Projections [6] available in the Charm++ framework). A time profile shows vertical bars for each (consecutive) time interval of 100 us, activities executed by the program added across all the processors. The red (dark) colored "peaks" at the bottom correspond to the integration step, while the dominant blue (light) colored regions represent non-bonded computations. The pink and purple (dark at the top) shade appearing in a thin layer every 4 steps represent the PME computation. One can notice that (a) time steps "bleed" into each other, overlapping in time. This is due to lack of a barrier, and especially useful when running on platforms where the OS noise may introduce significant jitter. While some processor may be slowed down on each step, the overall impact on execution is relatively small. (b) PME computations, which have multiple phases of large latencies, are completely overlapped with non-bonded computations.

2.2 Topology Sensitive Load Balancers

NAMD uses a measurement based load balancing. Initially the cells and computes are assigned to processors using a simple algorithm. After a user-specified number of time-steps, the runtime system turns on auto-instrumentation to measure the amount of work in each compute object. It then uses a greedy algorithm to assign computes to processors.

The application NAMD can be optimized to specific topologies on architectures where the

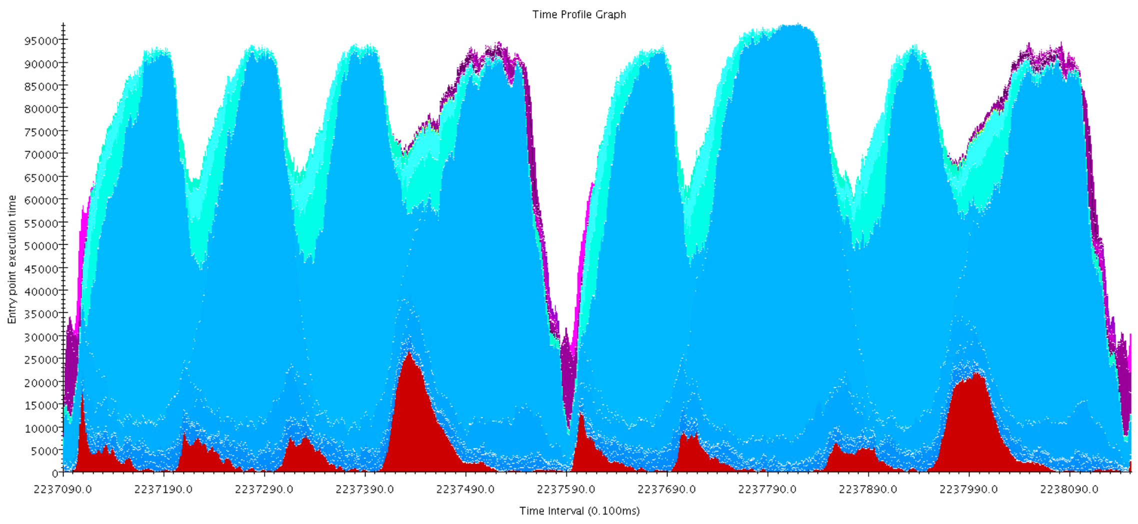


Figure 3: Time profile of ApoA1 on Blue Gene/L in Projections

topology information is available to the application. For example the Blue Gene/L machine has a torus interconnect for application message passing. The dimensions of the torus and the mapping of ranks to the torus is available through the personality data structure.

At application startup, the Charm++ runtime reads the personality structure to configure internal torus optimized map objects. As the periodic molecular systems are 3D Tori, we explored mapping the cells on the Blue Gene/L torus to improve the locality of cell to cell communication. We use an ORB scheme to map cells to the processors [7]. First the cell grid is split into two equally loaded partitions. The load of each cell is proportional to the number of atoms in that cell and the communication load of the cell. The processor partition is then split into two with the sizes of two sub-partitions corresponding to the sizes of the two cell sub-partitions. This is repeated recursively till every cell is allocated to a processor.

The above scheme enables locality optimizations for cell-to-cell communication. The Charm++ dynamic load-balancer places compute objects that calculate the interactions between cells near the processors which have the cell data. The load-balancer tries to allocate the compute on the least loaded processor that is within 8 hops of the midpoint of the two cells. We have observed that locality optimizations can significantly improve the performance of NAMD on Blue Gene/L.

3 Recent Optimizations

3.1 2D Decomposition of Particle Mesh Ewald

NAMD uses the Particle Mesh Ewald method [2] to compute long range Coulomb interactions. PME is based on real-to-complex 3D Fast Fourier Transforms, which require all-to-all communication but do not otherwise dominate the computation. NAMD has used a 1D decomposition for the FFT operations, which requires only a single transpose of the FFT grid and it is therefore the preferred algorithm with slower networks or small processor counts. Parallelism for the FFT in the 1D decomposition is limited to the number of planes in the grid, 108 processors for ApoA1. Since the message-driven execution model of Charm++ allows the small amount of FFT work to

Molecular System	**No. of atoms**	No. of signatures		Memory Footprint (MB)	
		Bonded Info	Non-bonded Info	Original	Current
IAPP	5570	102	117	0.290	0.022
DHFR (JAC)	23558	389	674	1.356	0.107
Lysozyme	39864	398	624	2.787	0.104
ApoaA1	92224	423	729	7.035	0.125
F1-ATPase	327506	737	1436	20.460	0.215
STMV	1066628	461	713	66.739	0.120
Bar Domain	1256653	481	838	97.731	0.128
Ribosome	2820530	1026	2024	159.137	0.304

Table 2: Number of Signatures and Comparison of Memory Usage for Static Information

be interleaved with the rest of the force calculation, NAMD can scale to thousands of processors even with the 1D decomposition. Still, we observed that this 1D decomposition limited scalability for large simulations on Blue Gene/L and other architectures.

We implemented a 2D decomposition for PME, where the FFT calculation is decomposed into thick pencils with 3 phases of computation and 2 phases of transpose communication. The FFT operation is computed by 3 arrays of objects in Charm++ with a different array for each of the dimensions. PME has 2 additional computation and communication phases that send grid data between the patches and the PME Charm++ objects. One of the advantages on the 2D decomposition is that the number of messages sent or received by any give processor is greatly reduced compared to the 1D decomposition for large simulations running on large numbers of processors.

3.2 Compression of Molecular Structure Data

The design of NAMD makes every effort to insulate the biophysicist from the details of the parallel decomposition. As such, the molecular structure is read from a single file on the head node and the data replicated across that machine. This structure assumes that each node of a parallel machine has sufficient memory to perform the simulation serially, as has historically been the case. The design of NAMD has, therefore, been based on using parallelism to gain additional processing power but not additional memory. Simulations now stretch into the millions of atoms, and future simulations have been proposed with 100 million atoms. At the same time, the design of highly scalable machines, such as Blue Gene/L, has begun to assume that the problem is completely distributed in memory and hence 512 MB or less per process is sufficient. This limited the simulations that could be run on Blue Gene/L to several hundred-thousand atoms. (This limitation could be partially alleviated on the Cray XT3 and other machines by adding memory to particular nodes, which were then reserved for the NAMD processes responsible for input, output, and load balancing.)

The molecular structure file read by NAMD describes the whole system information, including atoms' physical attributes, bonded structures (tuples of atoms such as bonds, angles, dihedrals . . .), etc. It is convenient to have this information available on every node, since when atoms move from one cell to another, i.e., very likely from one node to another, it would be complex and inefficient to migrate static data of arbitrary size as well. In order to reduce the memory footprint for this static information, we have developed a compression method that reduces memory usage by orders

	p1	p2	p3	p4
Reduction in L1 Cache Misses	7.45	2.76	4.18	3.12
Reduction in L2 Cache Misses	17.03	15.25	12.34	18.83
Reduction in L3 Cache Misses	1.57	2.06	2.09	2.82

Table 3: Reduction in Cache Misses Due to Structure Compression for ApoA1 on 4 Processors of SGI Altix

of magnitude while slightly improving performance due to reduced cache misses. The method leverages the common building blocks (amino acids, nucleic acids, lipds, water, etc.) from which large biomolecular simulations are assembled. Each tuple that pertains to a given atom is converted from absolute to relative atom indices, and the set of tuples for a given atom is definined as its *signature*. Atoms playing identical roles in the molecular structure will have identical signatures, and each unique signature need only be stored once, while the signature index for each atom is trivially distributed. Extracting signatures requires loading the entire structure, which is done on a separate large-memory workstation, producing a compressed molecular structure file which is read by NAMD on the small-memory Blue Gene/L.

Using structure compression we can now run the largest production simulations yet attempted, such as the ribosome with 2.8 million atoms, on Blue Gene/L. Table 2 shows the number of signatures for bonded static information and nonbonded static information respectively across a bunch of atom systems, and the resulting memory reduction ratio. The number of signatures increases only with the number of unique proteins in a simulation, and hence ApoA1 and STMV have similar numbers of signatures despite an order of magnitude difference in atom count. Thus, the technique is scalable to even simulations of 100-million atoms. We also observe a reduction in cache misses, as shown in Table 3, resulting in slightly better performance.

3.3 Flexible Decomposition

As discussed in Section 2, when we approach smaller atoms to processor ratios, we decompose the cells further into smaller cells to achieve higher parallelism. We can do so along each axis. So the size of the cell can be reduced by k along x, y, and/or z where k can be 1, 2 or 3 (rarely). These decompositions are called 1-Away, 2-Away and so on.

We give names to these further decompositions along the different axes: when splitting only along x, it is called 2-AwayX, when along X and Y, 2-AwayXY and finally 2-AwayXYZ. The important decision to be made at runtime is whch decomposition to choose for a given atoms to processors ratio for a given machine. We must create enough objects so that we have expressed the required concurrency, without undue increase in the overhead of too many small messages. The specific points where we switch from 1-Away to 2-AwayX or 2-AwayXY to 2-AwayXYZ depends also on the machine apart from the atoms to processors ratio.

NAMD is flexible in the sense that the user can choose these parameters for a particular run (taking into consideration the benchmark, number of processors and the machine). On the other hand, to save the common user from the burden of setting these parameters, the switch from one decomposition to a finer one has been automated. Once it is observed that we do not have enough computes for parallelization, the cells are divided along one more dimension.

Nodes	w/o (ms/step)	with (ms/step)
512	6.02	5.01
1024	3.48	2.96
2048	2.97	2.25

Table 4: Comparison of NAMD with and without using spanning trees, ApoA1 benchmark, Cray XT3

3.4 Communication Optimizations

Multicast was found to be a performance bottleneck in NAMD. Multicasts were previously treated as individual sends, paying the overhead of message copying and allocation. This is reasonable, especially on a small number of processors, since almost every processor is an originator of a multicast, and nothing much is gained by using a spanning tree. However, while running on a large number of processors, this imposes significant overhead on the multicast root processor (those processors that have home cells) when it needs to send a large number of messages. This was optimized by using a spanning tree implementation for the multicast operation to distribute the send overhead along the spanning trees node processors. At each level of a spanning tree, an intermediate node forwards the message to all its spanning children using an optimized send function to avoid message copying.

Although using spanning trees potentially improves parallel performance by offloading send overhead to intermediate nodes in the trees, it may incur higher latency when the depth of the spanning tree is large. Using the Projections performance analysis tool, we found that the multicast message may be delayed at the intermediate nodes when the nodes are busy doing computation [7]. To prevent this from happening, we exploited the *immediate messages* supported in Charm++. Immediate messages in Charm++, when supported on a platform such as Blue Gene/L, bypass the message-queue, and are processed immediately when a message arrives (instead of waiting for computation to finish). Using immediate messages for the multicast spanning trees helps to improve the responsiveness of forwarding messages by the intermediate nodes. Further, we use only a two-level spanning tree to reduce critical paths.

When load balancing re-assigns compute objects to processors, the spanning trees have to be updated according to the newly assigned multicast processors. The way spanning trees are constructed can effect the load balancing decision since it needs to take into account of the extra communication overhead associated with the intermediate nodes of the spanning tree. However, load balancing and the construction of the spanning trees are in different phases, therefore load balancing strategy when making compute-to-processor mapping decisions, does not have the information ready about the new spanning tree. Worse, load balancing and the spanning tree construction depend on each other — spanning trees can only be determined after load balancing decisions are made. Our current solution is to preserve the way spanning trees are built across load balancing steps as much as possible. Such persistent spanning tree helps load balancer evaluate the communication overhead. In future, we plan to have a separate load balancing step which makes decisions solely based on the multicast cost. It fine-tunes load balance by taking compute objects away from those overloaded processors with intermediate nodes. Further, we expect that support from lower-level communication layers (such as that used in Blue Matter) and/or hardware support for multiple concurrent multicasts will reduce the load (and therefore the importance) of the intermediate nodes

Molecular System	**No. of atoms**	**Cutoff (Å)**	**Simulation Box**	**Time step (fs)**
IAPP	5570	12	$46.70 \times 40.28 \times 29.18$	2
DHFR (JAC)	23558	9	$62.23 \times 62.23 \times 62.23$	1
Lysozyme	39864	12	$73.92 \times 73.92 \times 73.92$	1.5
ApoA1	92224	12	$108.86 \times 108.86 \times 77.76$	1
F1-ATPase	327506	12	$178.30 \times 131.54 \times 132.36$	1
STMV	1066628	12	$216.83 \times 216.83 \times 216.83$	1
Bar Domain	1256653	12	$195.40 \times 453.70 \times 144.00$	1
Ribosome	2820530	12	$264.02 \times 332.36 \times 309.04$	1

Table 5: Benchmarks and their simulation used for running NAMD in this paper

of the spanning trees.

With these optimization of the multicast operations in NAMD, parallel performance was significantly improved as shown in Table 4.

4 Performance Results

A highly scalable and portable application, NAMD has been tested on a variety of platforms for several benchmarks. The platforms vary from small-memory and moderate speed machines like the Blue Gene/L to faster processor machines like the Cray XT3. The results in this paper range from benchmarks as small as IAPP with 5570 atoms to Ribosome which has 2.8 million atoms. With the recent techniques for parallelization and memory optimization, NAMD has shown excellent performance in different regimes. Table 5 lists the various molecular systems and their simulation details which were used for the performance numbers in this paper.

A description of the various architectures on which the results were obtained for this paper follows:

- **IBM's Blue Gene/L:** The Blue Gene/L machine at IBM T J Watson (referred to as the "Watson Blue Gene/L" or **BG/L** in this paper) has 20480 nodes. Each node contains two 700 MHz PowerPC 440 cores and has 512 MB of memory shared between the two. The nodes on BG/L are connected into a 3D torus. We used the Watson BG/L for most of our runs. For our 32,678 processors runs, we used the BG/L at Lawrence Livermore National Laboratory (LLNL) which had 65,536 compute nodes.

- **Cray's XT3: BigBen** at PittsBurgh Supercomputing Center has 2068 compute nodes each of which has two 2.6 GHz AMD Opteron processors. The two processors on a node share 2 GB of memory. The nodes are connected into a 3D torus by a custom C-star interconnect.

- **IBM's P655+ and P690 cluster:** This cluster at the San Diego Supercomputing Center is called **DataStar** and has 272 (8-way) P655+ and 6 (32-way) P690 compute nodes. The 1.5 GHz 8-way nodes have 16 GB, the 1.7 GHz 8-way nodes have 32 GB, while four 32-way nodes have 128 GB of memory. DataStar has a nominal theoretical peak performance of 15.6 TFlops.

- **Dell's Linux Cluster:** This machine at the Texas Advanced Computing Center is known as

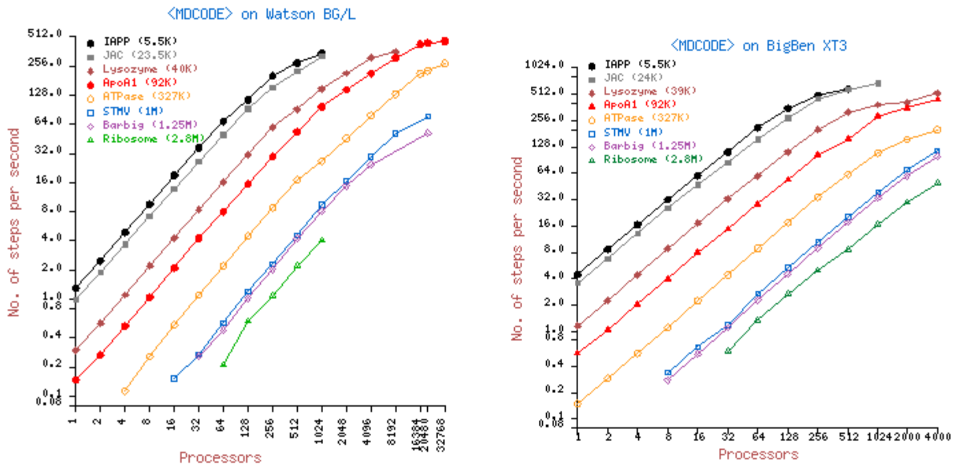


Figure 4: Performance of NAMD on IBM Blue Gene/L and Cray XT3

	ApoA1	**F1-ATPase**	**STMV**	**Bar D.**	**Ribosome**
#Procs on BG/L	32768	32768	20480	20480	1024
IBM BG/L (TFLOPS)	1.53	2.62	3.13	2.38	0.22
#Procs on Cray XT3	4000	4000	4000	4000	4000
Cray XT3 (TFLOPS)	1.46	1.94	4.72	4.60	2.64

Table 6: Floating Point Performance of a few benchmarks on BG/L and Cray

LoneStar and has 1300 nodes. Each node contains two 2.66 GHz Xeon dual-core processors with 4GB of memory each. This cluster has an Infiniband interconnect connecting the nodes.

4.1 Performance Results

Figure 4 shows the performance of the eight representative benchmarks from Table 5 on Watson BG/L and BigBen. We present performance numbers for the larger benchmarks upto 16,384 processors. We just got enough time to run ApoA1 and ATPase on up to 32,768 processors on the LLNL Blue Gene/L and they scale well.

Figure 5 shows the performance of a subset of the benchmarks on two new machines, the SDSC DataStar and TACC LoneStar clusters. We ran our application on these machines only recently and the numbers are without any specific optimizations for these machines. This exemplifies that even without any tuning for specific architectures, NAMD performs quite well on newer platforms. There is still scope to optimize NAMD on DataStar and LoneStar which will improve the performance even more. The same data is included in Tables **??** and **??** for BG/L and Cray XT3.

Table 6 shows the floating point operations per second for some of the benchmarks calculated for BG/L and Cray XT3.

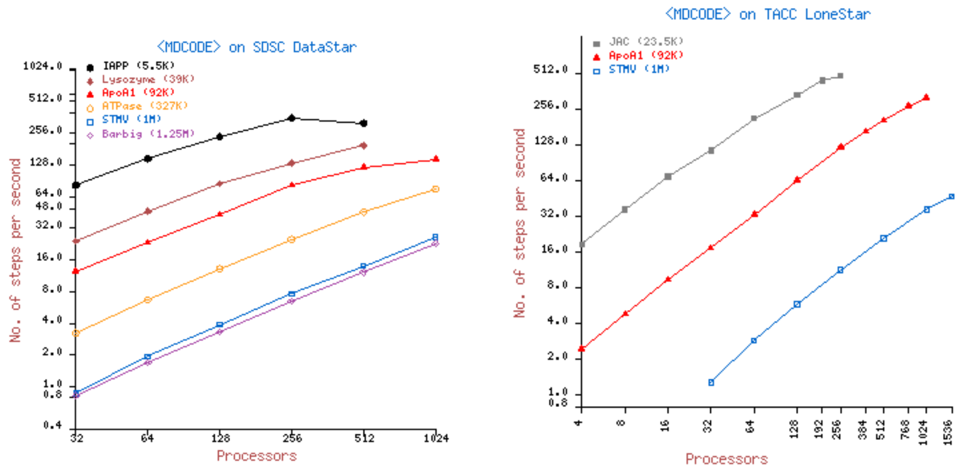


Figure 5: Performance of NAMD on SDSC DataStar and TACC LoneStar

4.2 Comparative Evaluation

Blue Matter is a program developed and optimized for the Blue Gene/L machine, and has achieved excellent scalability. Table 7 compares its performance with NAMD. On Blue Gene/L, each node has two processors, and one has an options of using the second processor as a computational process (this is called the “virtual node” mode) or just using it as a co-processor. Blue Matter essentially uses the second processor as a co-processor, but off-loads some computations (PME) to it. NAMD can use both processors for computation, or just use one processor on each node. The co-processor mode also has the advantage that the communication hardware is not shared with the second processor. As the table shows, NAMD is about 1.8 times faster than Blue Matter on 1024 processors, even if we were to restrict it to use only one processor per node. If it uses both processors (in co-processor node), on the same hardware configuration of 1024 node, it performs about 2.5 times faster than Blue Matter. However, it should be noted that NAMD performs a PME computation every 4 steps, whereas Blue Matter does it every step. We believe that the numerical algorithm used in NAMD ensures that the multiple time-stepping scheme does not lose accuracy.

The advantage of NAMD continues through a larger number of processors, but the percentage difference shrinks. On 16k nodes, it performs better than NAMD in co-processor mode. We don’t yet have the data-point for the 16k nodes in co-processor mode (which requires almost the entire machine). Performance data for Blue matter for larger molecular systems was not available.

We compare NAMD with a new program, *Desmond*, using a machine similar to that used in their SC’06 paper, although the TACC machine has slighly faster processors (2.6 GHz vs 2.4 Ghz on their cluster). Both machines use infiniband. Our performance is better until 512 processors although Desmond does better at 1024. This in spite of the (surmised) fact that Desmond used single precision arithmetic, and thereby exploited SSE instructions on the processor.

No of cores	**512**	**1024**	**2048**	**4096**	**8192**	**16384**	**32768**
Blue Matter		38.42	18.95	9.97	5.39	3.14	2.09
NAMD CO mode	17.47	9.56	5.84	3.86	2.91	2.14	
NAMD VN mode	17.98	9.82	6.26	4.34	3.06	2.36	2.11

Table 7: Comparison of NAMD and Blue Matter. The numbers are benchmark times for ApoA1 in milliseconds.

No of cores	**8**	**16**	**32**	**64**	**128**	**256**	**512**	**1024**
Desmond	256.8	126.8	64.3	33.5	18.2	9.4	5.2	3.0
NAMD on TACC	207.74	106.89	58.06	30.18	15.61	8.23	4.92	3.16

Table 8: Comparison of NAMD and Desmond. The numbers are benchmark times for ApoA1 in milliseconds.

4.3 Performance Analysis

4.3.1 Communication Volume

To enable an application to scale to a large number of processors its communication overhead must be scalable too. Our decomposition algorithm ensures that the total number bytes going on the network is bounded. However, the communication volume of NAMD depends on the dynamic load-balancer which tries to place compute objects on lightly loaded processors. Table 1 shows the communication statistics of the ApoA1 benchmark from 512 nodes to 20480 nodes of Blue Gene/L.

Observe that the communication volume rises from 47.3 MB on 512 nodes to about 254 MB on 20480 nodes. This is a total increase of 5.4 times while the number of processors has increased 40 times and the bisection bandwidth increased 8 times. The 2-Away options are responsible for keeping the communication volume bounded. It is observed that at 1024 nodes the 2-AwayXY option actually reduces the total communication volume resulting in the lower time-step.

As the communication volume is bounded, we do observe decreasing step times up to 20480 nodes with a speedup of over 3000. This can be attributed to the increasing communication to computation ratio as we approach higher number of processors. We believe that it is an achievement for a general purpose application to achieve this performance on Blue Gene/L with a relatively small problem like ApoA1.

4.3.2 Scaling Analysis

To analyze the bottlenecks in the most efficient parallelization of NAMD, we used the the summary data provided by Projections which gives detailed information about the time elapsed in the execution of each function in the program and also the time for which each processor is idle and doing no work. This information was collected on the Cray XT3 machine at PSC and the Blue Gene/L machine at for 1 to 4096 processors. The information is shown in Figure 6.

To simplify the figures, functions involved in communication have been grouped together and so have been other functions which perform the bonded and non-bonded work. There are two entries which correspond to the work involved in the integration and communication involved with the cells. The first observation is that the time for which processors are idle (waiting for

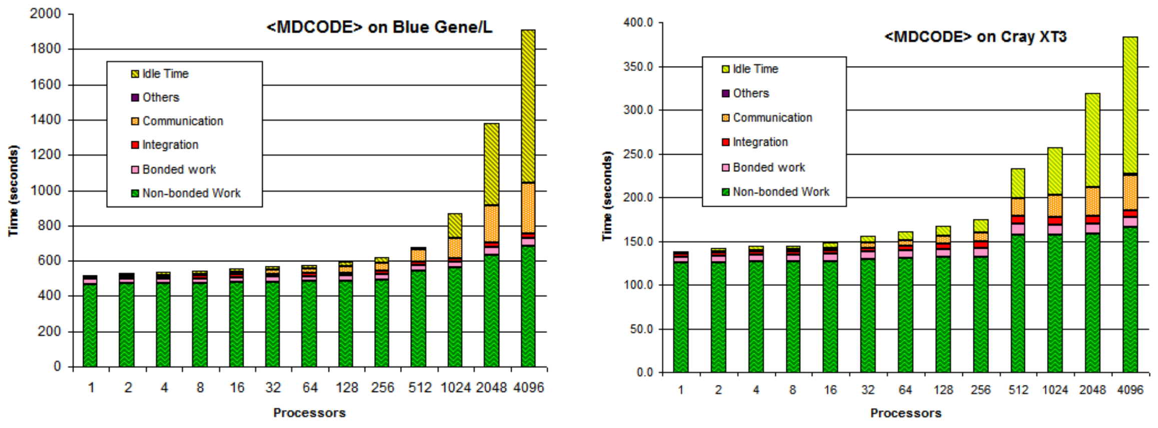


Figure 6: Percentage increase of different parts of NAMD with increase in number of processors (ApoA1)

computation to complete on which they are dependent) rises rapidly beyond 256 processors which is due to load imbalance. This is expected from the efficiency analysis we get from the benchmark times. There is a jump in the non-bonded work from 256 to 512 which can be attributed to the change in the decomposition strategy from 2-AwayX to 2-AwayXY at that point which doubles the number of cells. The other important slowdown is because of the doubling of communication as we move across the processors. This reason for this was discussed in the previous section and we think that there might be some room for improvement there.

Similar facts are seen on both BigBen and Watson BG/L. Observe that the scale of y-axis on the two sub-figures is different. So the difference is that idle time is not as significant on BG/L as it is on Cray because of slower processors.

5 Summary and Future Work

We described a highly scalable and portable parallel molecular dynamics program for biophysical simulations. Several of its features and algorithmic choices that are responsible for this scalability were described, and performance analysis of these features was presented. Some recent optimizations aimed at enhancing scalability were presented. We then demonstrated portability, scalability over a number of processors, and scalability across molecular systems ranging from small (5.5k atoms) to large (2.8M atoms), via performance data on multiple parallel machines, going up to 32k processors. We expect to have performance data on 12,500 processor Cray XT4 in the final version, once we have access to this new machine.

We believe that with these results NAMD has established itself as a high performance program that can be used at any of the national supercomputing centers; It is already a code that is routinely used and trusted by biophysicist. With the new techniques presented, it is ready for the next generation of parallel machines that will be broadly deployed in the coming years. Further optimizations for long term multi-petascale future machines include a larger reduction in memory footprint to accommodate the NSF 100M atom simulation, parallel I/O (especially to avoid memory bottleneck), and improved load balancers.

References

[1] K. J. Bowers, E. Chow, H. Xu, R. O. Dror, M. P. Eastwood, B. A. Gregersen, J. L. Klepeis, I. Kolossvary, M. A. Moraes, F. D. Sacerdoti, J. K. Salmon, Y. Shan, and D. E. Shaw. Scalable algorithms for molecular dynamics simulations on commodity clusters. In *SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, New York, NY, USA, 2006. ACM Press.

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

[3] B. Fitch, R. Germain, M. Mendell, J. Pitera, M. Pitman, A. Rayshubskiy, Y. Sham, F. Suits, W. Swope, T. Ward, Y. Zhestkov, and R. Zhou. Blue Matter, an application framework for molecular simulation on Blue Gene. *Journal of Parallel and Distributed Computing*, 63:759–773, 2003.

[4] B. G. Fitch, A. Rayshubskiy, M. Eleftheriou, T. J. C. Ward, M. Giampapa, and M. C. Pitman. Blue matter: Approaching the limits of concurrency for classical molecular dynamics. In *SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, New York, NY, USA, 2006. ACM Press.

[5] 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. *Journal of Computational Physics*, 1998.

[6] L. V. Kale, G. Zheng, C. W. Lee, and S. Kumar. Scaling applications to massively parallel machines using projections performance analysis tool. Technical Report 04-05, Parallel Programming Laboratory, Department of Computer Science, University of Illinois at Urbana-Champaign, March 2004.

[7] S. Kumar, C. Huang, G. Zheng, E. Bohm, A. Bhatele, J. C. Phillips, H. Yu, and L. V. Kalé. Scalable Molecular Dynamics with NAMD on Blue Gene/L. *IBM Journal of Research and Development: Applications of Massively Parallel Systems (to appear)*, 2007.

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

[9] M. Snir. A note on n-body computations with cutoffs. *Theory of Computing Systems*, 37:295–318, 2004.