Overcoming Scaling Challenges in Biomolecular Simulations across Multiple Platforms

Abhinav Bhatelé*, Sameer Kumar‡, Chao Mei*, James C. Phillips§, Gengbin Zheng*, Laxmikant V. Kalé*

* Department of Computer Science
University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA
E-mail: {bhatele2, chaomei2, gzheng, kale}@uiuc.edu

‡ IBM T. J. Watson Research Center
Yorktown Heights, NY 10598, USA
E-mail: sameerk@us.ibm.com

§ Beckman Institute
University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA
E-mail: jim@ks.uiuc.edu

Abstract

NAMD† is a portable parallel application for biomolecular simulations. NAMD pioneered the use of hybrid spatial and force decomposition, a technique now used by most scalable programs for biomolecular simulations, including Blue Matter and Desmond developed by IBM and D. E. Shaw respectively. NAMD has been developed using Charm++ and benefits from its adaptive communication-computation overlap and dynamic load balancing. This paper focuses on new scalability challenges in biomolecular simulations: using much larger machines and simulating molecular systems with millions of atoms. We describe new techniques developed to overcome these challenges. Since our approach involves automatic adaptive runtime optimizations, one interesting issue involves dealing with harmful interaction between multiple adaptive strategies. NAMD runs on a wide variety of platforms, ranging from commodity clusters to supercomputers. It also scales to large machines: we present results for up to 65,536 processors on IBM's Blue Gene/L and 8,192 processors on Cray XT3/XT4. In addition, we present performance results on NCSA's Abe, SDSC's DataStar and TACC's LoneStar cluster, to demonstrate efficient portability. We also compare NAMD with Desmond and Blue Matter.

†NAMD stands for NAnoscale Molecular Dynamics

1 Introduction

Molecular Dynamics (MD) simulations of biomolecular systems constitute an important technique for understanding 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), which means a million to billion time-steps. In contrast to billion-atom simulations needed in material science, biological systems require simulations involving only thousands of atoms to a few million atoms, as illustrated in Fig. 1. The main challenge here is to take a computation that takes only a few seconds per time-step and run it on thousands of processors effectively.

Biomolecular systems are fixed and limited in size (in terms of the number of atoms being simulated). Even so, the size of the desired simulations has been increasing in the past few years. Although molecular systems with tens of thousands of atoms still remain the mainstay, a few simulations using multi-million atom systems are being pursued. These constitute a new set of challenges for biomolecular simulations. The other challenge is the emergence of ever-larger machines. Several machines with over hundred

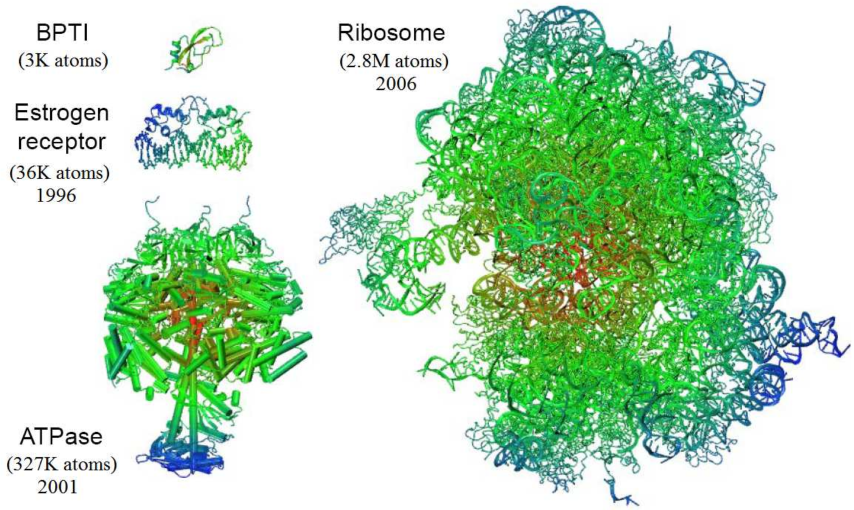

Figure 1. The size of biomolecular system that can be studied through all-atom simulation has increased exponentially in size, from BPTI (bovine pancreatic trypsin inhibitor), to the estrogen receptor and F_1-ATPase to ribosome. Atom counts include solvent, not shown for better visualization.

thousand processors and petaFLOPS peak performance are planned for the near future. NSF recently announced plans to deploy a machine with *sustained* petaFLOPS performance by 2011 and provided a biomolecular simulation benchmark with 100 million 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 focuses on our efforts to meet these challenges and the techniques we have recently developed to this end. One of these involves dealing with an interesting interplay between two adaptive techniques – load balancing and adaptive construction of spanning trees – that may be instructive to researchers who use adaptive runtime strategies. Another involves reducing memory footprint without negative performance impact.

Another 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. 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, such as NAMD, that performs well across a range of architectures is therefore desirable.

These techniques and resultant performance data are new since our IPDPS'06 paper [11]. An upcoming paper in IBM Journal Res. Dev. [12] will focus on optimizations specific to Blue Gene/L, whereas an upcoming book chapter [14] summarizes performance data with a focus on science. The techniques in this paper are not described and analyzed in these publications.

Finally, two new programs have emerged since our last publication: *Blue Matter* [6, 7] from IBM that runs on the Blue Gene/L machine, and *Desmond* [2] from D. E. Shaw that runs on their Opteron-Infiniband cluster. We include performance comparisons with these, for the first time. (In SC'06, both programs compared their performance with NAMD. This paper includes an updated response from the NAMD developers). Both Desmond and Blue Matter are highly scalable programs and use variations of the idea developed in NAMD (in our 1998 paper [9]) of computing forces between atoms belonging to two processors (potentially) on a third processor. NAMD assigns these force computations based on a measurement-based dynamic load balancing strategy (see Sec. 2.2), whereas the other approaches use fixed methods to assign them. We show that NAMD's performance is comparable with these programs even with the specialized processors each runs on. Moreover NAMD runs on a much wider range of machines and molecular systems not demonstrated by these programs.

We showcase our performance on different machines including Cray XT3/XT4 (up to 8,192 processors), Blue

Gene/L (up to 65,536 processors), TACC's Lonestar (up to 1,024 processors) and SDSC's DataStar cluster (up to 1,024 processors) on molecular systems ranging in size from 5,570 atoms to 2.8 million atoms. We also present an analysis that identifies new bottlenecks that must be overcome to enhance the performance with a large number of processors.

We first describe the basic parallelization methodology adopted by NAMD. We then describe a series of optimizations, alternative algorithms, and performance trade-offs that were developed to enhance the scalability of NAMD. For many of these techniques, we provide analysis that may be of use to MD developers as well as other parallel application developers. Finally, we showcase the performance of NAMD for different architectures and compare it with other MD programs.

2 Background: Parallel Structure of NAMD

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 naïve evaluation of these forces takes 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 (PME) 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. PME is the generally accepted method for long-range electrostatics in biomolecular simulations, for good reasons. The $O(N \log N)$ complexity applies only to the FFT, which in practice is a vanishing fraction of the runtime even for very large systems. The bulk of the calculation is $O(N)$. In addition, PME produces smooth forces that conserve energy well at lower accuracy. Achieving energy conservation with multipole methods (some of which are $O(N)$ [8]) requires much higher accuracy than dictated by the physics of the simulation. Thus, PME has a significant performance advantage over multipole methods for biomolecular simulations.

Prior to [9], parallel biomolecular simulation programs used either atom decomposition, spatial decomposition, or force decomposition for parallelization (for a good survey, see Plimpton et. al [13]). 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 [6, 7], the neutral territory and midpoint territory methods of Desmond [2], and those proposed by Snir [15] use variations of such a hybrid strategy.

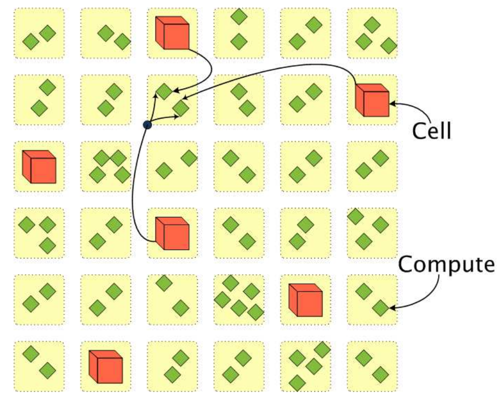

Figure 2. Placement of cells and computes on a 2D mesh of processors

NAMD decomposes atoms into boxes called "cells" (see Fig. 2). 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. The size d_i of a cell along each dimension i is such that for some integer m, B_i/m is just greater than d_{min}. For example, if $r_c = 12$ Å and $margin = 4$ Å, $d_{min} = 16$ Å, the cells should be of size $16 \times 16 \times 16$ Å. However, this is not the right choice if the simulation box is $108.86 \times 108.86 \times 77.76$ Å, as is the case in ApoLipoprotein-A1 (ApoA1) simulation (see Table 5). Since the simulation box along X dimension is 108.86 Å, one must pick $d_i = 108.86/6 = 18.15$ Å as the size along X axis for the cell. (And the size of a cell will be $18.15 \times 18.15 \times 19.44$ Å). 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") whose 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. The fact that these compute objects are assigned to processors by a dynamic load balancer (Sec. 2.2) gives NAMD 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, Y or Z 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,

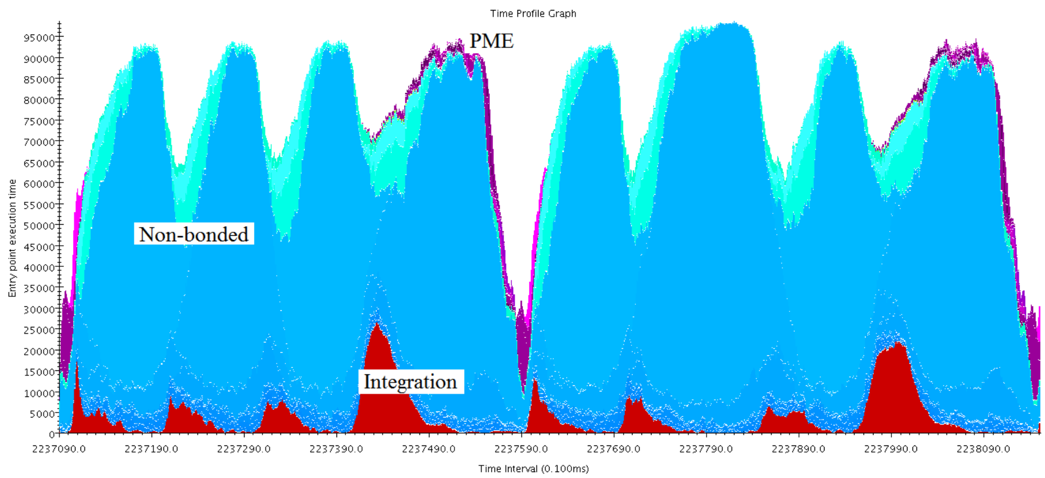

Figure 3. Time profile for ApoA1 on 1k processors of Blue Gene/L (with PME) in Projections

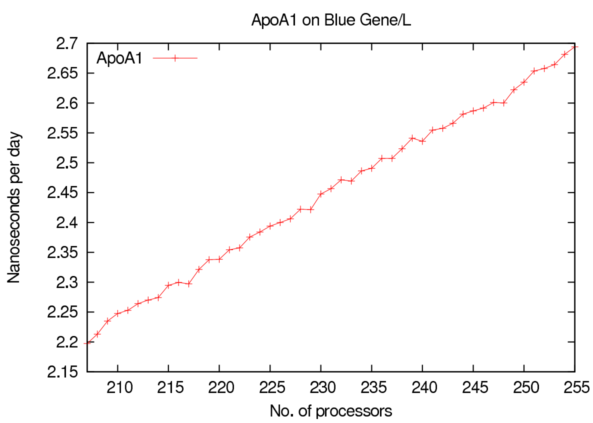

Figure 4. NAMD **performs well on any given number of processors (Plot shows ApoA1 running on Blue Gene/L, with PME on a range of processor counts varying from 207 to 255)**

this decomposition is called 2-Away-X, 2-Away-XY or 2-Away-XYZ etc. depending on which dimension uses $k = 2$. The choice of which decomposition to use for a particular run is decided by the program depending on the atoms-to-processor ratio and other machine-dependent heuristics. NAMD also gives the user flexibility to choose the decomposition for certain scenarios where the automatic choices do not give the best results.

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. Fig. 4 shows the performance of the simulation of ApoA1 on varying numbers of Blue Gene/L (BG/L) 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.

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, Fig. 3 shows a time profile of a simulation of ApoA1 on 1024 processors of BG/L (This figure was obtained by using the performance analysis tool Projections [10] 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 force 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

Processors	512	1024	1024	2048	4096	8192	8192	16384
Decomposition	X	X	XY	XY	XY	XY	XYZ	XYZ
No. of Messages	4797	7577	13370	22458	29591	35580	79285	104469
Avg Msgs. per processor	10	8	13	11	8	5	10	7
Max Msgs. per processor	20	31	28	45	54	68	59	88
Message Size (bytes)	9850	9850	4761	4761	4761	4761	2303	2303
Comm. Volume (MB)	47.3	74.6	63.7	107	141	169	182	241
Atoms per cell	296	296	136	136	136	136	60	60
Time step (ms)	17.77	11.6	9.73	5.84	3.85	3.2	2.73	2.14

Table 1. Communication Statistics for ApoA1 running on IBM's Blue Gene/L (without PME)

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 Dynamic Load Balancing

A challenging aspect of parallelizing biomolecular simulations is that a heterogeneous collection of computational work, consisting of bonded and direct electrostatic force evaluation, PME and integration of equations of motion, has to be distributed to a large number of processors. This is facilitated by measurement-based load balancing in NAMD. 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 load balancer assigns computes 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 multicast of coordinates to computes and 10 to return forces). Table 1 shows the number of messages (in the non-PME portion of the computation) as a function of number of processors for the ApoA1 simulation on BG/L. The number of cells and the decomposition used is also shown. The load balancer and the use of spanning trees for multicast (Sec. 3.1) 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. 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). Thus, per-message overhead is a more important consideration and hence we believe fewer-but-larger messages to be a better tradeoff.

NAMD can be optimized to specific topologies on architectures where the topology information is available to the application. For example the BG/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 a *personality data structure*. At application startup, the CHARM++ runtime reads this data structure [1]. As the periodic molecular systems are 3D Tori, we explored mapping the cells on the BG/L torus to improve the locality of cell to cell communication. We used an ORB scheme to map cells to the processors [12]. 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 a few hops of the midpoint of the two cells. We have observed that locality optimizations can significantly improve the performance of NAMD on BG/L.

3 Scaling Challenges and Techniques

Since the last paper at IPDPS [11], we have faced several scaling challenges. Emergence of massively parallel machines with tens of thousands of processors is one. Needs of biophysicists to run larger and larger simulations with millions of atoms is another. Hence it became imperative to analyze the challenges and find techniques to scale million-atom systems to tens of thousands of processors. This sec-

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

Table 2. Comparison of running NAMD with and without spanning trees (ApoA1 on Cray XT3, without PME)

tion discusses the techniques which were used to overcome these challenges and improve scaling of NAMD over the last few years.

3.1 Interaction of Adaptive Runtime Techniques

Multicasts in NAMD were previously treated as individual sends, paying the overhead of message copying and allocation. This is reasonable on a small number of processors, since almost every processor is an originator of a multicast and not much is gained by using spanning trees (STs) for the multicast. However, when running on a large number of processors, this imposes a significant overhead on the multicast root processors (which have home cells) when it needs to send a large number of messages. Table 1 shows that though the average number of messages (10) per processor is small, the maximum can be as high as 88. This makes a few processors bottlenecks on the critical path. To remove this bottleneck, a spanning tree implementation for the multicast operation was used to distribute the send overhead among the spanning tree node processors. At each level of a ST, an intermediate node forwards the message to all its children using an optimized send function to avoid message copying.

However, solving one problem unearthed another. Using Projections, we found that the multicast message gets delayed at the intermediate nodes when the nodes are busy doing computation. To prevent this from happening, we exploited *immediate messages* supported in CHARM++. Immediate messages in CHARM++, on a platform such as BG/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 intermediate nodes in forwarding messages [12].

Recently, it was noticed that even immediate messages did not improve the performance as expected. Again using Projections, we noticed that processors with multiple intermediate nodes were heavily overloaded. The reason is as follows: STs can only be created after the load balancing step. So, when the load balancer re-assigns compute objects to processors, it has no knowledge of the new STs. On the other hand, the spanning tree strategy does not know about the new load balancing decisions and hence it does not have any information about the current load. Moreover since the STs for different cells are created in a distributed fashion, multiple intermediate nodes end up on the same processor. This is a situation where two adaptive runtime techniques are working in tandem and need to interact to take effective decisions. Our solution is to preserve the way STs are built across load balancing steps as much as possible. Such persistent STs helps the load balancer evaluate the communication overhead. For the STs, the solution is to create the STs in a centralized fashion to avoid placing too many nodes on a single processor. With these optimizations of the multicast operations in NAMD, parallel performance was significantly improved as shown in Table 2.

A further step is to unite these two techniques into a single phase. We should do a load balancing step and using its decisions, we should create the STs. Then we should update the loads of processors which have been assigned intermediate nodes. With these updated loads we can do a final load balancing step and modify the created STs to take the new decisions into account. In the future, 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 STs.

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. Hence, the molecular structure is read from a single file on the head processor and the data is replicated across the machine. This structure assumes that each processor of a parallel machine has sufficient memory to perform the simulation serially. But machines such as BG/L assume that the problem is completely distributed in memory and hence 512 MB or less per processor is provided. Simulations now stretch into millions of atoms, and future simulations have been proposed with 100 million atoms. The molecular structure of large systems whose memory footprint grows at least linearly with the number of atoms limited the NAMD simulations that could be run on BG/L to several hundred-thousand atoms. To overcome this limitation, it became necessary to reduce the memory footprint for NAMD simulations.

The molecular structure in NAMD describes the whole system including all atoms' physical attributes, bonded structures etc. To reduce the memory footprint for this static information we developed a compression method that reduces memory usage by orders of magnitude and slightly

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
ApoA1	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 3. Number of Signatures and Comparison of Memory Usage for Static Information

Item	Before Opt.	After Opt.	Reduction Rate (%)
Execution time (ms/step)	16.53	16.20	2.04
L1 Data Cache Misses (millions)	189.07	112.51	68.05
L2 Cache Misses (millions)	1.69	1.43	18.18
TLB Misses (millions)	0.30	0.24	25.00

Table 4. Reduction in cache misses and TLB misses due to structure compression for ApoA1 running (with PME) on 128 processors of CrayXT3 at PSC

improves performance due to reduced cache misses. The method leverages the similar structure of common building blocks (amino acids, nucleic acids, lipids, water etc.) from which large biomolecular simulations are assembled.

A given atom is always associated with a set of tuples. This set is defined as its *signature* from which its static information is obtained. Each tuple contains the information of a bonded structure this atom participates in. Originally, such information is encoded by using absolute atom indices while the compression technique changes it to relative atom indices. Therefore, atoms playing identical roles in the molecular structure have identical signatures and each unique signature needs to be stored only once. For example, the oxygen atom in one water molecule plays identical roles with any other oxygen atom in other water molecules. In other words, all those oxygen atoms have the same *signature*. Therefore, the memory footprint for those atoms' static information now is reduced to the memory footprint of a *signature*.

Extracting signatures of a molecule system is performed on a separate large-memory workstation since it requires loading the entire structure. The signature information is stored in a new molecular structure file. This new file is read by NAMD instead of the original molecular structure file. Table 3 shows the number of signatures for bonded and non-bonded 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. Hence, ApoA1 and STMV have similar numbers of signatures despite an order of magnitude difference in atom count. Thus, the technique is scalable to simulations of even 100-million atoms.

Using structure compression we can now run million-atom systems on BG/L with 512MB of memory per node, including the 1.25 million-atom STMV and the 2.8 million-atom Ribosome, the largest production simulations yet attempted. In addition, we have also observed slightly better performance for all systems due to this compression. For example, the memory optimized NAMD version is faster by 2.1% percent than the original version for ApoA1 running on 128 processors of Cray XT3. Using the CrayPat‡ performance analysis tool, we found this better performance resulted from a overall reduction in cache misses and TLB misses per simulation time-step, as shown in Table 4. Such an effect is expected since the memory footprint for static information is significantly reduced which now fits into the L2 cache and requires fewer memory pages.

3.3 2D Decomposition of Particle Mesh Ewald

NAMD uses the Particle Mesh Ewald method [4] 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

‡http://www.psc.edu/machines/cray/xt3/#craypat

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 parameters used for running NAMD in this paper

System Name	Location	No. of Nodes	Cores per Node	CPU Type	Clock Speed	Memory per Node	Type of Network
Blue Gene/L	IBM	20,480	2	PPC440	700 MHz	512 MB	Torus
	LLNL	65,536	2	PPC440	700 MHz	512 MB	Torus
Cray XT3/XT4	PSC	2,068	2	Opteron	2.6 GHz	2 GB	Torus
	ORNL	8,532	2/4	Opteron	2.6 GHz	2 GB	Torus
DataStar	SDSC	272	8	P655+	1.5 GHz	32 GB	Federation
		6	32	P690+	1.7 GHz	128 GB	
LoneStar	TACC	1,300	4	Xeon	2.66 GHz	8 GB	Infiniband
Abe	NCSA	1,200	8	Intel64	2.33 GHz	8 GB	Infiniband

Table 6. Specifications of the parallel systems used for the runs

Processors	1D (ms/step)	2D (ms/step)
2048	7.04	5.84
4096	5.05	3.85
8192	5.01	2.73

Table 7. Comparison of 1D and 2D decompositions for FFT (ApoA1 on Blue Gene/L, with PME)

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 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 BG/L and other architectures.

PME is calculated using 3D FFTs [5] in Blue Matter. We implemented a 2D decomposition for PME in NAMD, 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 dimension. 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 given processor is greatly reduced compared to the 1D decomposition for large simulations running on large numbers of processors. Table 7 shows the advantages of using 2D decomposition over 1D decomposition.

Similar to 2-Away decomposition choices, NAMD can automatically choose between 1D and 2D decomposition depending on the benchmark, number of processors and other heuristics. This choice can also be overridden by the user. Hence, NAMD's design provides a runtime system which is capable of taking intelligent adaptive runtime decisions to choose the best algorithm/parameters for a particular benchmark-machine-number of processors combination.

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 frequency processors like Blue Gene/L to faster processors like Cray XT3. The results in this paper range from benchmarks as small as IAPP with 5570 atoms to Ribosome

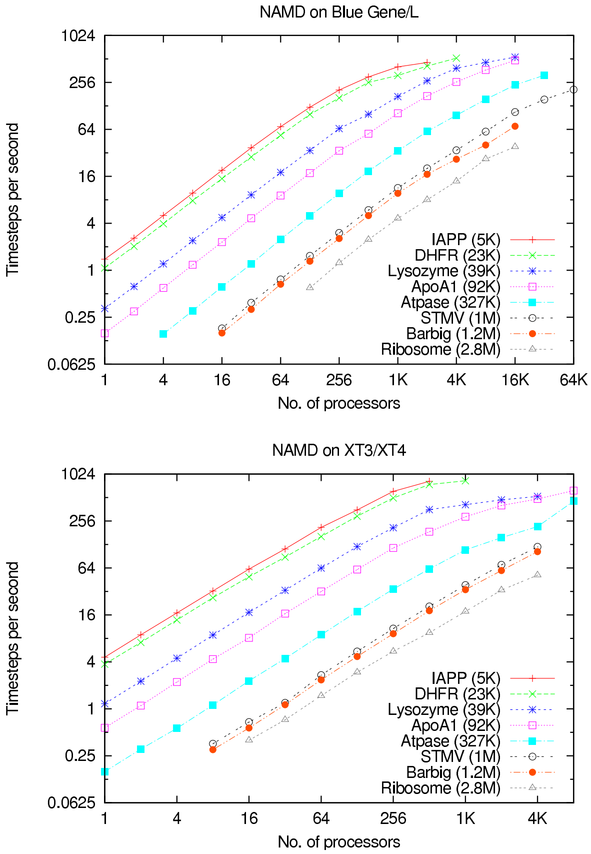

Figure 5. Performance of NAMD **on IBM BG/L and Cray XT3/XT4**

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 is shown in Table 6.

Fig. 5 shows the performance of the eight representative benchmarks from Table 5 on BG/L and XT3/XT4. On BG/L, we present performance numbers for ATPase up to 32K and STMV up to 64K processors. We just had enough time to run these two benchmarks on the larger 64K LLNL BG/L. Using different adaptive runtime strategies and algorithms discussed in the previous sections, we show good scaling for all benchmarks on this machine. For Cray, we present numbers for ApoA1 and ATPase on 8,192 processors on the Jaguar XT4 at ORNL. With the faster processors we achieve a lowest time-step of 1.19 ms/step for DHFR which corresponds to 72 ns of simulation per day. It is an achievement that for IAPP running on XT3, we can simulate

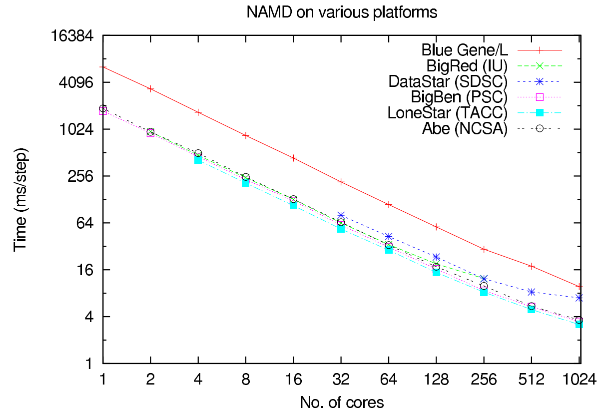

Figure 6. Performance of NAMD **on multiple parallel machines (Benchmark: ApoA1)**

140 ns/day with a 2 fs time-step. Table 8 shows the floating point operations per second (FLOPS) for some of the benchmarks calculated for BG/L and Cray XT3. They were calculated by using the Perfsuite performance suite on a Xeon cluster (Tungsten) at NCSA. For STMV, we do nearly 8.5 TeraFLOPS on 64K processors of BG/L. We did not have a chance to improve this number because we only have a limited access to the LLNL BG/L.

Fig. 6 shows the performance of the ApoA1 benchmark on a variety of machines including: SDSC's DataStar and TACC's LoneStar and NCSA's Abe. 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. We do not show performance beyond 1K processors because without further performance tuning for the respective machines, NAMD does not scale well (beyond 1K). There is still scope to optimize NAMD on these machines which will improve the performance even more.

5 Comparison with other MD codes

In this section, we compare NAMD's performance with two other MD codes, Blue Matter [6, 7] and Desmond [2]. Fig. 7 shows the performance for ApoA1 using NAMD on some representative machines, Blue Matter on BG/L and Desmond on their cluster. It is noteworthy that NAMD running on 4K processors of XT3 achieves better performance than Blue Matter on 32K processors of BG/L and similar to Desmond running on 2K processors of their Opteron machine.

Blue Matter is a program developed and optimized for the BG/L machine, and has achieved excellent scalability.

Benchmark	ApoA1	F1-ATPase	STMV	Bar D.	Ribosome
#Procs on BG/L	16,384	32,768	65,536	16,384	16,384
IBM BG/L (TFLOPS)	1.54	3.03	8.44	3.22	2.07
#Procs on Cray XT3	8,192	8,192	4,096	4,096	4,096
Cray XT3/XT4 (TFLOPS)	2.05	4.41	4.87	4.77	2.78

Table 8. Floating Point Performance of a few benchmarks on BG/L and XT3/XT4 in TeraFLOPS

Number of Nodes	512	1024	2048	4096	8192	16384
Blue Matter (2 pes / node)	38.42	18.95	9.97	5.39	3.14	2.09
NAMD *CO mode (1 pe / node)*	16.83	9.73	5.8	3.78	2.71	2.04
NAMD *VN mode (2 pes / node)*	9.82	6.26	4.06	3.06	2.29	2.11
NAMD *CO mode (PME every step)*	19.59	11.42	7.48	5.52	4.2	3.46
NAMD *VN mode (PME every step)*	11.99	9.99	5.62	5.3	3.7	-

Table 9. Comparison of benchmark times (in ms/step, ApoA1) for NAMD **and Blue Matter**

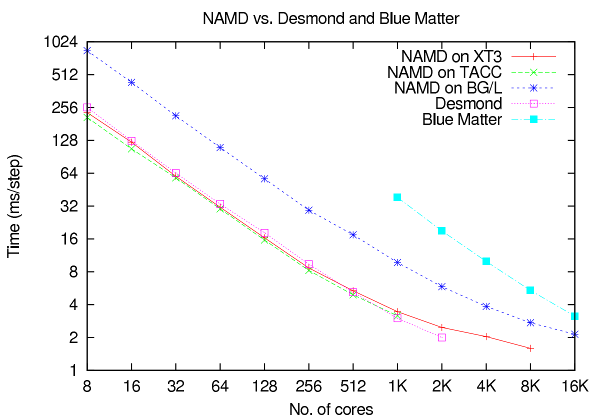

Figure 7. Performance comparison of NAMD **with Desmond and Blue Matter for ApoA1**

Table 9 compares its performance with NAMD. On BG/L, each node has two processors and one has an option of using the second processor as a computational process (this is called the "virtual node" (VN) 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 (CO) mode also has the advantage that the communication hardware is not shared with the second processor. As the table shows, NAMD is about 1.95 times faster than Blue Matter on 1,024 nodes even if we were to restrict it to use only one processor per node. If it uses both processors (in VN mode), on the same hardware configuration of 1,024 nodes, it performs about 3.03 times faster than Blue Matter. NAMD scales all the way up to 16K nodes both in CO and VN mode. This has been achieved by improving the performance of NAMD by about 200% (since [11] which was the basis for Blue Matter's comparison in [7]) by removing PME and communication bottlenecks. 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. For a fair comparison, benchmark times for performing the PME computation every step are also included. Though Blue Matter has better scaling compared to NAMD for PME every step runs, the performance of NAMD is still comparable to that of Blue Matter.

We compare NAMD with Desmond using a machine similar to that used in their SC'06 paper, although the TACC machine has slightly faster Xeon 5100 processors (2.6 GHz) compared to the Sun Fire V20z processors (2.4 GHz) on their cluster. Both machines use Infiniband as the interconnect. We used two cores per node for running NAMD on LoneStar to compare fairly with the two cores per node runs of Desmond. Our performance for ApoA1 and DHFR is better than Desmond all the way up to 2048 cores. This in spite of the (surmised) fact that Desmond used single precision arithmetic, and thereby exploited SSE instructions on the processor. Since Desmond is proprietary software, so we could not run it on the same machine as NAMD. Although the comparison is not apples-to-apples (since processor speeds and cache sizes for the two machines are different), it still demonstrates that NAMD's performance capabilities are comparable with Desmond's. Moreover, we could not reproduce the anomalous performance of NAMD cited in [2] on any similar machine.

No of Cores	8	16	32	64	128	256	512	1024	2048
Desmond ApoA1	256.8	126.8	64.3	33.5	18.2	9.4	5.2	3.0	2.0
NAMD *ApoA1*	199.25	104.96	50.69	26.49	13.39	7.12	4.21	2.53	1.94
Desmond DHFR	41.4	21.0	11.5	6.3	3.7	2.0	1.4		
NAMD *DHFR*	27.25	14.99	8.09	4.31	2.37	1.5	1.12	1.03	

Table 10. Comparison of benchmark times (ms/step) for NAMD (running on 2.6 GHz Opterons) and Desmond (running on 2.4 GHz Opterons)

6 Future Work

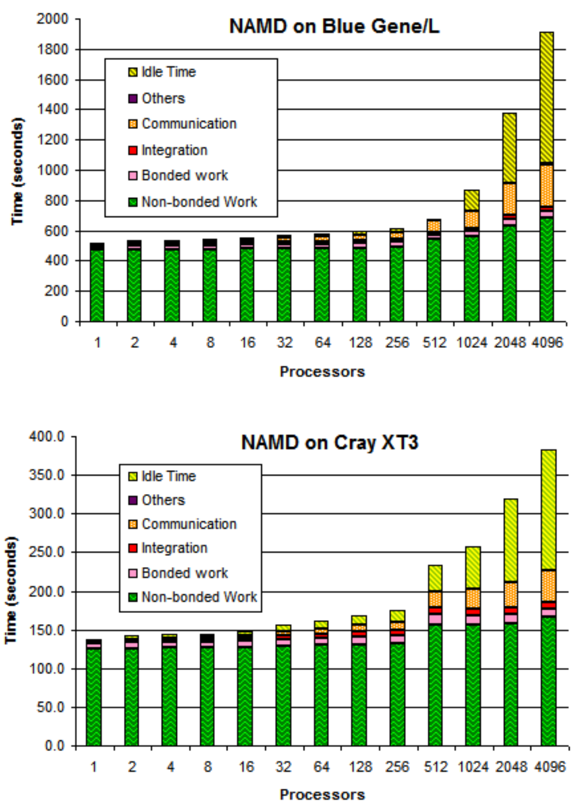

Figure 8. Percentage increase of different parts of NAMD with increase in number of processors (ApoA1 on BG/L and XT3, without PME)

The needs of biomolecular modeling community require us to pursue strong scaling i.e. we must strive to scale the same molecular system to an ever larger number of processors. We have demonstrated very good scalability with the techniques described in this paper, but challenges remain, especially if we have to exploit the petascale machines for the same problems. To analyze if this is feasible and what avenues are open for further optimizations, we carried out a performance study of scaling that we summarize below.

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. This data, collected on the Cray XT3 machine at PSC and the BG/L machine at IBM for 1 to 4,096 processors, is shown in Fig. 8.

To simplify the figures, functions involved in similar or related activities are grouped together. The first observation is that the idle time rises rapidly beyond 256 processors. This is mostly due to load imbalance, based on further analysis. One of the next challenges is then to develop load balancers that attain better performance while not spending much time or memory in the load balancing strategy itself. Also, 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-Away-X to 2-Away-XY at that point which doubles the number of cells. Since the essential computational work involved in non-bonded force evaluation does not increase, we believe that this increase can be reduced by controlling the overhead in scheduling a larger number of objects. The other important slowdown is because of the increase in communication overhead as we move across the processors. Looking back at Table 1 we see that from 512 to 16K processors, the computation time per processor should ideally decrease 32-fold but does not. This can be attributed in part to the communication volume (per processor) decreasing only by a factor of about 10. Although some of this is inevitable with finer-grained parallelization, there might be some room for improvement.

7 Summary

The need for strong scaling of fixed-size molecular systems to machines with ever-increasing number of processors has created new challenges for biomolecular simulations. Further, some of the systems being studied now include several million atoms. We presented techniques we developed or used, to overcome these challenges. These included dealing with interaction among two of our adaptive strategies: generation of spanning trees and load balancing. We also presented new techniques for reducing the memory footprint of NAMD to accommodate large benchmarks, including NSF-specified 100M atom benchmark for the "Track-1" sustained petascale machine. 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 64K processors for the largest simulations. We also presented a performance study that identifies new bottlenecks that must be overcome to enhance the performance with a large number of processors, when the ratio of atoms to processors falls below 50.

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 a code which is routinely used and trusted by the 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.

Acknowledgments

This work was supported in part by a DOE Grant B341494 funded by the Center for Simulation of Advanced Rockets and a NIH Grant PHS 5 P41 RR05969-04 for Molecular Dynamics. This research was supported in part by NSF through TeraGrid resources [3] provided by NCSA and PSC through grants ASC050039N and ASC050040N. We thank Shawn T. Brown from PSC for helping us with benchmark runs on BigBen. We also thank Fred Mintzer and Glenn Martyna from IBM for access and assistance in running on the Watson Blue Gene/L. We thank Wayne Pfeiffer and Tom Spelce for benchmarking NAMD on the SDSC DataStar and LLNL Blue Gene/L respectively. Figure 1 was prepared by Marcos Sotomayor as part of a proposal to fund NAMD development.

References

[1] A. Bhatele. Application-specific topology-aware mapping and load balancing for three-dimensional torus topologies. Master's thesis, Dept. of Computer Science, University of Illinois, 2007. http://charm.cs.uiuc.edu/papers/BhateleMSThesis07.shtml.

[2] 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.

[3] C. Catlett and et. al. TeraGrid: Analysis of Organization, System Architecture, and Middleware Enabling New Types of Applications. In L. Grandinetti, editor, *HPC and Grids in Action*, Amsterdam, 2007. IOS Press.

[4] 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.

[5] M. Eleftheriou, B. G. Fitch, A. Rayshubskiy, T. J. C. Ward, and R. S. Germain. Scalable framework for 3D FFTs on the Blue Gene/L supercomputer: Implementation and early performance measurements. *IBM Journal of Research and Development*, 49(2/3), 2005.

[6] 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.

[7] 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.

[8] L. Greengard and V. I. Rokhlin. A fast algorithm for particle simulations. *Journal of Computational Physics*, 73, 1987.

[9] 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.

[10] L. V. Kale, G. Zheng, C. W. Lee, and S. Kumar. Scaling Applications to Massively Parallel Machines Using Projections Performance Analysis Tool. In *Future Generation Computer Systems Special Issue on: Large-Scale System Performance Modeling and Analysis*, volume 22, pages 347–358, February 2006.

[11] S. Kumar, C. Huang, G. Almasi, and L. V. Kalé. Achieving strong scaling with NAMD on Blue Gene/L. In *Proceedings of IEEE International Parallel and Distributed Processing Symposium 2006*, April 2006.

[12] 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)*, 52(1/2), 2007.

[13] 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.

[14] K. Schulten, J. C. Phillips, L. V. Kale, and A. Bhatele. Biomolecular modeling in the era of petascale computing. In D. Bader, editor, *Petascale Computing: Algorithms and Applications*, pages 165–181. Chapman & Hall / CRC Press, 2008.

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