Scalable molecular dynamics with NAMD on the IBM Blue Gene/L system

S. Kumar
C. Huang
G. Zheng
E. Bohm
A. Bhatele
J. C. Phillips
H. Yu
L. V. Kalé

NAMD (nanoscale molecular dynamics) is a production molecular dynamics (MD) application for biomolecular simulations that include assemblages of proteins, cell membranes, and water molecules. In a biomolecular simulation, the problem size is fixed and a large number of iterations must be executed in order to understand interesting biological phenomena. Hence, we need MD applications to scale to thousands of processors, even though the individual timestep on one processor is quite small. NAMD has demonstrated its performance on several parallel computer architectures. In this paper, we present various compiler optimization techniques that use single-instruction, multiple-data (SIMD) instructions to obtain good sequential performance with NAMD on the embedded IBM PowerPC® 440 processor core. We also present several techniques to scale the NAMD application to 20,480 nodes of the IBM Blue Gene/L™ (BG/L) system. These techniques include topology-specific optimizations to localize communication, new messaging protocols that are optimized for the BG/L torus, topology-aware load balancing, and overlap of computation and communication. We also present performance results of various molecular systems with sizes ranging from 5,570 to 327,506 atoms.

Introduction

With a greater understanding of the functioning of biological systems, the importance of biomolecular simulations has increased significantly. More than 43,000 structures are publicly available from the Protein Data Bank (*www.pdb.org*). The availability of such structures has enabled researchers to explore the relationship between structure and function through simulations that are now based on a firm experimental foundation. However, for most proteins, scientists are aware only of amino-acid sequences. Thus, molecular simulations are needed to predict the detailed three-dimensional (3D) structure of proteins through the process of simulated protein folding.

These kinds of molecular simulations present several computational challenges. Because hydrogen atoms in a biomolecule vibrate with a period of approximately 10 fs (femtoseconds), the computational timestep must be about 1 fs for stable and accurate integration. However, the phenomenon of interest may occur only at a scale of microseconds. Several nanoseconds of simulation may be required even to allow a protein to relax into the nearest low-energy state. Some phenomena can be studied by observing the behavior of the molecule over tens of nanoseconds. Given an initial state, we can force the simulation through interesting paths of behavior. However, even with computational shortcuts, we are still faced with the problem of simulating several million to a few billion timesteps.

For most molecular dynamics (MD) calculations, the number of atoms in a particular biologically interesting configuration is fixed. For example, if we want to study a particular *aquaporin* (a protein that straddles a cell membrane and allows water to cross the membrane), our simulation would involve one aquaporin tetramer, a reasonably sized patch of cell membrane in which to embed it, and a sufficient quantity of water molecules around the structure. This kind of study may comprise a

0018-8646/08/$5.00 © 2008 IBM

few hundred thousand atoms. Our simulations with a fixed problem size (e.g., a fixed number of atoms) are in contrast to the study of continuous phenomena, such as weather prediction, in which we can simply increase the resolution to exploit a computer with many processors. Of course, we can study increasingly large molecules as machines with more processors become available, but the size of such molecular assemblies of interest is normally limited and we still need to study a particular system to understand its behavior. The key challenge is to quickly simulate a large number of timesteps.

For some problems, multiple independent or weakly coupled (e.g., replica-exchange method [1]) simulations may be used to increase sampling of molecular configurations. However, for many simulations, artificial steering forces [2] are used to drive the molecule through a series of transitions [such as for the rotary mechanism of adenosine triphosphate (ATP) synthase]. Larger steering forces produce artifacts; thus, in order to enhance accuracy, simulations are performed using the smallest steering forces possible that provide results in a reasonable time. This suggests that longer simulations are necessary for accuracy. Larger molecules function through longer and more complex mechanisms, making even longer simulations necessary. Thus, larger simulations have an even greater need for computational performance than implied by their atom counts alone, and the challenge of reducing wall-clock (elapsed) time per step remains.

In our research, we consider the apolipoprotein-A1 (ApoA1) system, a model of a high-density lipoprotein (HDL) particle [3]. (HDL transports cholesterol in the bloodstream and is commonly referred to as the *good* cholesterol when measured by blood samples taken in a physician's office.) With 92,224 atoms and a mix of proteins, lipids, and water, ApoA1 is representative of modern moderately sized simulations. The serial (i.e., sequential) simulation of ApoA1 takes about 6.2 seconds per timestep on a single processor core of an IBM Blue Gene/L* (BG/L) system. The challenge is to replace the 6.2-second computation with one that runs efficiently on several thousand processors, with each timestep taking only a few milliseconds.

In this paper, we describe how this scaling was accomplished in the NAMD (nanoscale molecular dynamics) program running on the BG/L system. NAMD [4] is a parallel MD application developed at the University of Illinois at Urbana-Champaign (UIUC) as a collaborative project involving the Theoretical and Computational Biophysics Group (*www.ks.uiuc.edu*) and the Computer Science Department, which includes the Parallel Programming Laboratory (*charm.cs.uiuc.edu*). NAMD uses an effective parallelization strategy that is a hybrid of spatial decomposition and force decomposition. This is supported further by dynamic load-balancing capabilities of the Charm++ parallel programming system developed at the UIUC. This hybrid parallelization strategy has remained effective over the past 10 years. We begin by describing and reviewing this strategy. We then present a series of optimizations that include sequential optimizations targeted toward the IBM PowerPC* 440 (PPC440) processor cores used in the BG/L system, as well as parallel optimizations that include dynamic load balancing. We present performance data for molecular systems ranging from 5,570 to 327,506 atoms.

NAMD parallelization strategy

NAMD uses a hybrid strategy that combines spatial decomposition with force decomposition and couples it with the dynamic load-balancing framework of Charm++. The dominant computation in MD is that of nonbonded forces (i.e., electrostatic and van der Waals forces) between all pairs of atoms. The potential $O(n^2)$ all-pairs algorithm is optimized to $O(n \log n)$ complexity by using the notions of a cutoff radius r_c, and separation of computation of short-range and long-range forces. For each atom, the nonbonded forces due to atoms within r_c are calculated explicitly. The long-range forces due to the atoms outside this radius are calculated using an $O(n \log n)$ particle-mesh Ewald algorithm. Even with this splitting, 90% of the computation cost is due to explicit calculation of nonbonded forces within the cutoff radius.

Initial attempts at parallel MD simulations in the field of biophysics were made using existing sequential codes. We showed in [4] that many such strategies were not scalable, in the sense of the isoefficiency metric for scalability [5]. (The isoefficiency metric indicates how well a parallel workload scales while maintaining a fixed efficiency.) In particular, the communication-to-computation ratio of many of these schemes rises with increasing numbers of processors, and in a way that does not allow weak scaling. In other words, even if we were to increase the number of atoms, the communication-to-computation ratio would not improve. A pure force-decomposition scheme, such as that in [6, 7], also suffers from this limitation.

We also showed in [4] that spatial decomposition does not have this problem. We presented an even more effective formulation that combines spatial decomposition and force decomposition that generates additional parallelism without increasing communication. In this formulation, the simulation space is divided into cubic boxes (called *patches* in NAMD). The size of each cube b is chosen based on the cutoff distance r_c. Let $B = r_c + r_H + m$, where r_H is twice the maximum length of a bond to a hydrogen atom. The variable m, the margin, is twice the distance that atoms may move

without migration between patches being necessary. Variable b along each dimension is chosen as B/k. Typically, k is either 1 or 2 (although we have experimented with $k = 3$). With $k = 1$ (also referred to as *one-away decomposition*), there are about 400–700 atoms per cube. With $k = 2$, this decreases to 50–75 atoms. In order to provide intermediate granularities, we support noncubic patches. For example, each dimension can be either B or $B/2$, for example. With $k = 1$, only atoms in neighboring patches have to interact (and there are 27 interactions in which each patch participates). With $k = 2$, interactions involve 125 cubes that are two-away from each other in the coordinate space.

An innovation in NAMD 2.0 [4] was its use of a kind of force decomposition on top of this spatial decomposition; for each pair of interacting patches, NAMD creates a *force-computation object* (referred to as a *compute object* or *compute* for brevity). With $k = 1$, this leads to 14 times more compute objects than the number of patches. These compute objects are then assigned to processors under the control of a dynamic load balancer in Charm++ (see below). Compared to spatial decomposition by itself, this strategy also eliminates duplicate computation of forces, by exploiting Newton's third law. The parallelization of NAMD with patch and compute objects is shown in **Figure 1**.

More recent proposals for scalable MD [8–10] use the basic strategy of hybrid decomposition, which originated in NAMD 2.0. The proposals differ in how the force computations are assigned to processors, that is, via either static (but topology-sensitive) schemes [8, 9] or the Blue Matter MD scheme [10] that allocates work on the basis of the number of atoms. We believe that the original scheme is at least as good as or better than these schemes because of its ability to take the dynamic processor load into account (assuming that the load balancer performs well).

Charm++ is a C++-based parallel programming system in which the programmer decomposes the work into a large number of interacting message-driven objects called *chares*. The objects may be organized into multiple indexed collections, known as *chare arrays*. The ontology of the programmer does not include processors but only the objects. Multiple objects can be and typically are assigned to a single processor. The execution on each processor is controlled by a scheduler, which selects an available message, identifies the chare for which it is destined, allows the chare to process the message, and repeats. The Charm++ adaptive runtime system can reassign objects to processors during a run and handles all the bookkeeping associated with such migrations automatically. It measures computational loads of individual objects and tracks communication between pairs of objects. On the basis of these measurements, the load balancer can reassign objects accurately to improve load balance and to decrease communication.

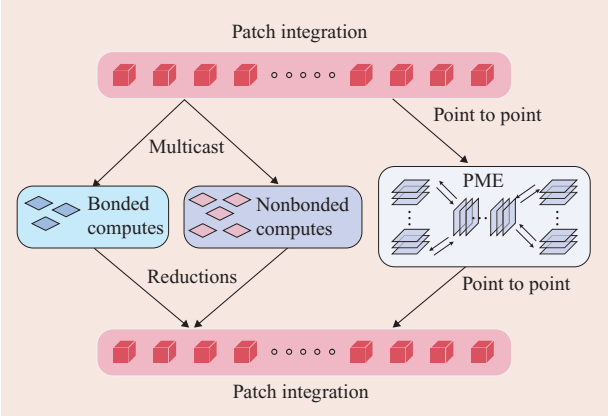

Figure 1

Parallelization of NAMD computation. The "bonded computes" refer to that portion of the code involved with computing forces due to atomic bonds, and the "nonbonded computes" refer to the portion of the code that computes electrostatic forces.

Blue Gene/L optimizations

Improving sequential performance on the PowerPC 440 core

The BG/L system is based on the embedded PPC440 core. A significant effort was required to achieve good sequential performance on the BG/L embedded core. Because of aliasing constraints in the NAMD inner loop, the IBM XL (extensible language) compiler was unable to generate optimized code. As a result, the inner force compute loops were not effectively software pipelined. We eliminated the aliasing constraints by inserting `#pragma disjoint` directives in the compute loop to enable the generation of software-pipelined object code. In some instances, we manually unrolled and pipelined the loop to obtain the best performance. (When we use the phrase *unroll the loop*, we mean the process in which the instructions that are called in multiple iterations of the loop are combined into a single iteration.) We observed that the bonded code in NAMD had several stack temporaries because it was using C++ operator overloading. (*Bonded code* refers to that part of the code that computes the forces due to atomic bonds.) The stack temporaries introduced expensive loads, stores, and pipeline stalls. We reported this to the IBM XL compiler team and obtained a fix that is now available in XLC version 8.0. The net result of the above optimizations was to more than double serial performance on the PPC440. The optimizations also helped other PowerPC Architecture*

Table 1 Single-processor timestep (seconds) on various PowerPC Architectures for ApoAl.

Architecture	*NAMD 2.5*	*NAMD 2.7 prerelease*
PowerPC 440 700 MHz	14.0	6.60
PowerPC 440d 700 MHz	14.1	6.17
PowerPC 970 2.0 GHz	3.90	1.87
POWER4* 1.5 GHz	4.80	2.25

platforms. **Table 1** compares the performance of NAMD version 2.5 with the prerelease version of NAMD 2.7. (The BG/L systen optimizations in NAMD were added after the release of NAMD version 2.5.)

The PPC440 core is enhanced with an additional floating-point unit called the *double floating-point unit* (double FPU) [11]. In order to take advantage of the double FPU, the addresses of loads and stores must be aligned to 16 bytes. We had to pad the force vector and other structures (which were originally 24 bytes) to have a 32-byte alignment, in order to make use of the double FPU. The force computation in NAMD requires that the X, Y, and Z dimensions be computed. However, the SIMD instructions can parallelize only the X and Y dimensions, and the computation of the Z dimension is not parallelized by SIMD instructions. This restricts the achievable speedup from the double FPU to be about 33%.

The actual performance improvement with the SIMD optimizations is only about 7%, as shown in Table 1. We are working on further optimizing the SIMD version of NAMD. A reason for lower 440d performance may be a cache miss in the force-compute loop. (*440d* refers to the SIMD extensions for the PPC440 core.) This loop uses interpolation tables for the three independent terms in the energy potential (electrostatic plus Lennard–Jones r^{-12} and r^{-6}), thus eliminating reciprocal-square-root and *erfc* (complementary error function) computations. The interpolation table is quite large and of the order of several hundred cache lines. Many PowerPC Architecture implementations have relatively small level 1 (L1) caches (32 KB on the BG/L system). This may lead to cache misses in the inner compute loop of NAMD. Because the access pattern to the interpolation table is quite irregular, even the level 2 (L2) prefetch unit on the Blue Gene/L chip may not be effective in this case.

We are exploring new computational algorithms that may have more SIMD computation but would require a smaller interpolation table. We also plan to take advantage of the PowerPC reciprocal-square-root approximation instruction in order to further reduce the number of entries in the interpolation table. We hope these optimizations will further improve the performance of NAMD on the BG/L platform as well as other PowerPC Architecture platforms.

Topology mapping

The BG/L supercomputer has a torus interconnection network [12] for application data exchange. Because a torus interconnection network has limited bisection bandwidth, localizing communication results in better application performance. For NAMD, this fact makes careful mapping of patches to processors critical for achieving strong scaling on the BG/L system. The patches in NAMD are allocated to processors using an orthogonal recursive bisection (ORB) scheme [13] to map the patch objects to the BG/L torus. The dimensions of the BG/L torus depend on the size of the processor partition, while the dimensions of the patch torus depend on the size of the problem. Both the processor and patch tori are typically not cubic. Thus, first, the axes of the patch and processor tori are sorted, and then the largest dimensions of the processor torus are matched with the corresponding dimensions of the patch torus. For example, when we map a $13 \times 6 \times 4$ patch torus to an $8 \times 32 \times 16$ processor torus, we obtain the following axis map: $X_{\text{procesor}} = Z_{\text{patch}}$, $Y_{\text{processor}} = X_{\text{patch}}$, $Z_{\text{processor}} = Y_{\text{patch}}$. Next, we rotate the patch grid so that its dimensions match the processor grid using the above computed axis map. Once this is done, we can use ORB (as described below) to allocate patches to processors.

The ORB scheme splits patches along the longest dimension and then computes the total load of each of the two partitions. The load generation function takes into account the patch computation, which depends on the number of atoms and the expected communication overhead of each patch. Next, the processor grid (torus) is halved to form two subgrids in which the size of each partition corresponds to the load of the patch partition. This is repeated recursively until we have one patch that is allocated to a random processor in the corresponding processor subgrid. **Figure 2** shows the mapping of patches and computes onto the 3D torus of the BG/L system.

Communication optimizations

We have developed a native Charm++ runtime system, optimized for the BG/L platform on top of the BG/L message layer [14]. We found the Charm++ Message Passing Interface (MPI) driver not suitable to make progress on the network because it called MPI Iprobe and MPI Test functions, thus introducing communication overheads. An optimized runtime also allowed us to explore the *adaptive eager* messaging protocol that does not require the ordering semantics of MPI.

The production MPI software [15] on the BG/L system has two protocols for point-to-point messages: *eager* and *rendezvous*. In the eager protocol, all packets are sent using the in-order deterministic routing scheme. The first packet matches the MPI-posted receives and the rest of the payload is copied into an application buffer. In the rendezvous protocol, first an RTS (request-to-send) packet is sent to the receiver. When the receiver is ready to receive, it sends a CTS (clear-to-send) packet back to the sender. Upon receiving the CTS, the sender sends the application buffer using adaptive routing.

Because the eager protocol uses deterministic routing, it can restrict the communication performance of the application. Even though the rendezvous protocol uses adaptive routing, it has a three-way handshake, which is not very effective in the Charm++ scenario. The three-way handshake restricts overlap of computation and communication. It is possible that the sender starts computing after sending the messages and cannot process the CTS packet as soon as it arrives. Moreover, the overhead of the RTS and CTS packets makes rendezvous less suitable for the most common NAMD messages, which are only a few kilobytes.

We present the adaptive eager protocol to optimize the scenario in which the application sends several short messages that are a few kilobytes. The adaptive eager protocol sends messages with adaptive routing while avoiding RTS and CTS packets. In this protocol, each processor keeps a system-wide connection list with one slot for each processor in the booted partition. Each packet must carry all the state information for the message. In MPI, this state would include the communicator and tag for the message, along with the source and the size. Thus, for an MPI implementation of the adaptive eager protocol, the tag and communicator would also have to be sent to match the incoming message with a list of posted receives. Hence, the bandwidth achievable with the adaptive eager protocol in MPI will be lower than that for eager or rendezvous. Moreover, only one message can be outstanding because messages must be matched in order.

Unlike in MPI, in Charm++ all messages are received as *unexpected* messages, so only the size of the message is needed in all arriving packets to allocate a buffer for the message. Fortunately, packets on the BG/L system have an 8-byte software header. We were able to pack the message size, source, packet offset, and a sequence number into the 8-byte software header, with each of those fields occupying 21, 18, 21, and 4 bits, respectively. After receiving a packet, the receiver looks up the connection list, and if a buffer for that sequence number has not been allocated, it makes a request of the application (e.g., the Charm++ runtime application) for a buffer to receive the message.

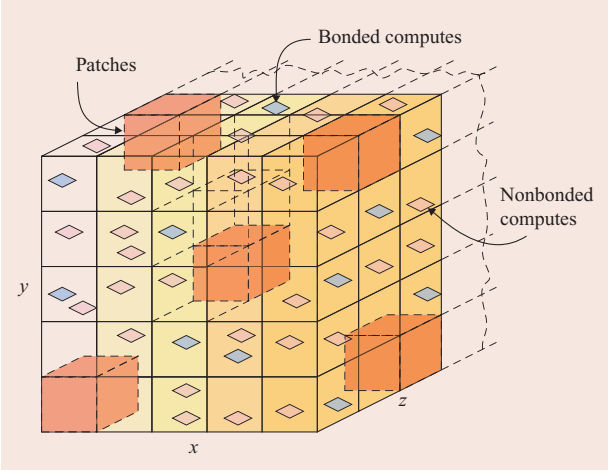

Figure 2

Mapping of patches and computes onto the BG/L torus.

As Charm++ does not require message ordering, we can allow several messages to be outstanding at a given time. The packets of these messages can arrive together and are distinguished by a sequence number. The maximum number of outstanding messages is determined by the number of bits allocated to the sequence number. This corresponds to 16 outstanding messages with a 4-bit sequence number. Once the receiver has received all of the 16 messages, it sends an acknowledgment back to the sender, which then sends the next set of 16 messages.

Dynamic load balancing

NAMD uses the Charm++ dynamic load-balancing framework [16, 17]. The patches are initially placed in a topology-optimized manner using the ORB scheme presented in the section on topology mapping. In order to achieve good computational and communication load balancing, the Charm++ runtime must record the most up-to-date application and system load information. The Charm++ runtime exploits a simple heuristic called *principle of persistence* [18] to automatically obtain load information. This principle simply exploits the fact that the object computation times and communication patterns (number and bytes of messages exchanged between each communicating pair of objects) tend to persist over time, which holds for MD simulations in which atoms move slowly. This heuristic makes it possible to instrument the application automatically at runtime and use the newly instrumented load information to predict the load in the near future. In NAMD, the load-balancing framework measures the computational load of all the objects along with the communication and background load of nonmigratable work on the

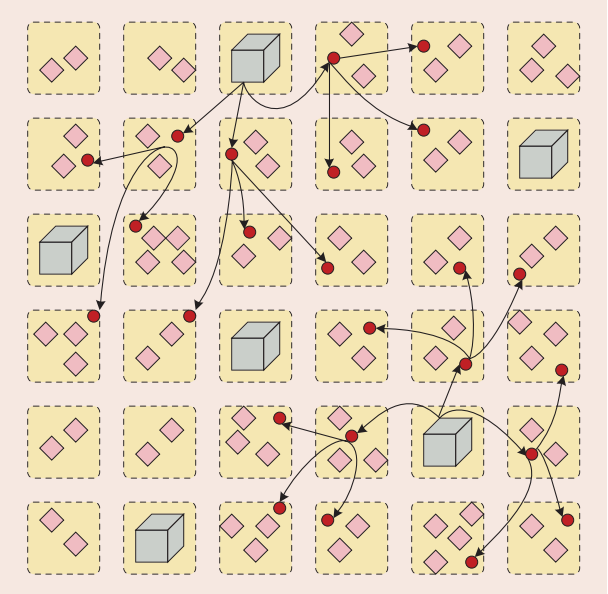

Figure 3

Spanning tree showing multicast origins (i.e., patches represented by cubes) and proxies (circles).

processors. The load statistics are provided as a parameter to a load-balancing strategy (i.e., algorithm) that computes the new object placements.

We use two load-balancing strategies in NAMD. The first scheme involves a comprehensive load-balancing strategy that assigns migratable work (mostly nonbonded force computation), ignoring the initial placement of such work. This comprehensive scheme is performed only once during a run of NAMD. The second load-balancing strategy involves a refinement strategy that moves just a few objects from overloaded processors to lightly loaded processors. The refinement scheme (i.e., procedure) is called periodically (every few thousand timesteps) to move compute objects to balance changes in processor load for atom migrations.

Both the comprehensive and the refinement strategies have topology optimizations built into them. In the comprehensive strategy, the load balancer first assigns all the compute objects to a max-heap. The strategy then picks the most overloaded compute object and assigns it to a processor on the basis of a greedy heuristic that takes into account the processor load, the communication history, the proximity of the processor on the BG/L torus to the patches whose interaction is being computed, and the number of destinations in the patch multicast. The size of the patch multicast depends on the number of proxies, which are destination processors that keep copies of the patch coordinate data for one or more local computes. The comprehensive strategy is biased by initial proxies placed on processors that are close to the patch processor on the BG/L torus. This strategy also favors the use of processors that are fewer than four hops from the midpoint of the patches whose interaction is being computed in the compute object.

In the refinement strategy, the overloaded processors are first allocated to a max-heap. The strategy involves a loop that picks the highest-loaded processors and then removes the overloaded objects in that processor to a lightly loaded accepting processor. This accepting processor is chosen using heuristics similar to those in the comprehensive scheme. The refinement strategy iterates until no overloaded processors exist with a load above a threshold. The refinement strategy favors lightly loaded processors within eight hops of the midpoint of the patches whose interaction is being computed.

Once the load-balancing strategy has finished reassigning compute objects, the Charm++ runtime application moves the objects to their new destinations. After the moves are finished, a new spanning tree is constructed for each patch to multicast its atom coordinate data. The spanning tree creation also ensures that no processor is overloaded with spanning tree intermediates from different patches. **Figure 3** shows patches and their multicast targets superimposed on a two-dimensional (2D) view of the physical processor topology. A two-level k-ary tree is generally used in NAMD, with the value of k close to ten.

Overlap of computation and communication

The BG/L system has two PPC440 cores on each node. However, it does not have a DMA (direct memory access) unit on the compute node. Ideally, one of the cores could serve as a communication coprocessor, but because of lack of cache coherence, the caches must be flushed for any communication between the cores. The overhead of cache flushing may limit the performance of the coprocessor mode because the messages in NAMD are relatively short.

In this paper, we present a technique that can overlap computation and communication in virtual node mode [19]. Each core has six normal-priority torus FIFOs (first in, first outs), and each of these FIFOs can store up to four packets. At a full link bandwidth of 175 MB/s, each FIFO would fill up in about 4,320 processor cycles. We have observed in NAMD that the achievable throughput due to network contention corresponds to only about two links, which implies that each FIFO would fill up on average every 12,960 cycles. We can have the cores compute for these 12,960 cycles and periodically make calls in order to drain network FIFOs and call the progress engine in the messaging software. In NAMD, the rate of progress is specified as a command line

parameter and can be tuned to the processor partition size and the benchmark.

The BG/L torus interconnect is a reliable network in which a packet is sent downstream only if there are resources available for it. The local resources of the packet are released when the receiver acknowledges the error-free reception of the packet. Hence, the reception FIFOs must be drained before they fill to capacity. If this is not done, packets will be trapped in intermediate buffers on the network. The progress calls in NAMD prevent this from happening, as network reception FIFOs are drained from the inner compute loops.

We had to develop infrastructure in the Charm++ runtime system in order to support such progress calls from application entry methods. We extended this runtime to support immediate messages, within the progress calls. With an immediate method, the handler for the message is called within the progress call, allowing the message to be forwarded to other processors. The NAMD coordinate multicast uses a spanning tree to multicast data to the destinations (see the section on dynamic load balancing). If an intermediate destination on the spanning tree is busy in a compute loop, the multicast messages will be delayed, resulting in poor performance. With immediate messages, the multicast data can be forwarded on the intermediate nodes within a few thousand processor cycles after the message has arrived. Similarly, immediate messages can also be used for the force reduction messages that are sent back to the patches.

Particle-mesh Ewald

NAMD uses the particle-mesh Ewald (PME) method [20] to compute the long-range interactions between the atoms. The PME method requires two 3D fast Fourier transforms (FFTs) to be computed. NAMD 2.6 used a 1D decomposition for the FFT operations. Because the 1D decomposition requires only a single transpose of the FFT grid, it is the preferred algorithm on clusters with slower networks and small numbers of processors. Parallelism for the FFT in the 1D decomposition is limited to the number of planes in the grid, and 108 processors for the ApoA1 benchmark. However, the message-driven execution model of Charm++ allows a small amount of FFT work to be interleaved with the rest of the force calculation, allowing NAMD to scale to thousands of processors even with the 1D decomposition. Nevertheless, we have observed that this 1D decomposition does not scale well on the BG/L system and many other architectures because of insufficient parallelism.

We implemented a 2D decomposition for the PME method, in which the FFT calculation is decomposed into thick pencils with three phases of computation and two phases of transpose communication. A thick pencil along the X dimension will contain all the FFT grid points in the X dimension, while the Y and Z dimension sizes would typically vary from one to four grid points. The FFT operation is computed by three arrays of chares in Charm++ with a different array for each of the three phases of transposes. At the limits of scalability, this operation is mainly dominated by communication overhead of small transpose messages. We used the real-to-complex optimization to reduce the computation and communication overhead of the FFT operation by a factor of 2.

In addition to the two FFT calculations, PME in NAMD has two computation and communication phases. These phases send grid data from the patches in NAMD to the PME force computation and FFT chares. The PME calculation begins with the computation of the charge grid by interpolating each atom to a charge grid typically of size $4 \times 4 \times 4$. The contribution of each atom is reduced locally. Next, the intersecting section of the charge grid is sent to the FFT thick-pencil chare along the Z dimension. The FFT thick pencils preform a forward 3D FFT followed by the Ewald calculation on the transformed grid in k space. Next, a backward 3D FFT is performed that computes the long-range forces that are sent back to the patches. The forces are then integrated to update atom positions and velocities in the next integrate phase.

One of the advantages in 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. Consider a typical situation in which each 16-Å patch contributes to a $24 \times 24 \times 24$ block of the FFT grid. For an $N \times N \times N$ patch grid, each slab of an N-slab 1D decomposition communicates with up to $2N^2$ patches, and each patch communicates with 24 slabs. For the same system, with a 2D decomposition, each thick pencil of $m \times m$ grid lines communicates with at most $16N$ patches (assuming $m < 16$), and each patch communicates with $(24/m + 1)^2$ pencils. Thus, the 2D decomposition has fewer messages to and from patches if $N > 8$ and $m > 7$, a simulation of roughly 200,000 atoms. Similarly, messages per processor are reduced for the FFT transposes for pencils larger than 2×2 grid lines.

Performance results

We used four different molecular systems to obtain performance benchmarks for NAMD on the BG/L system. The benchmarks include the 5,570-atom islet amyloid polypeptide (IAPP) system [21], the lysozyme in urea simulation [22] (39,864 atoms), ApoA1 (92,224 atoms), and the F_1-ATPase system (327,506 atoms).

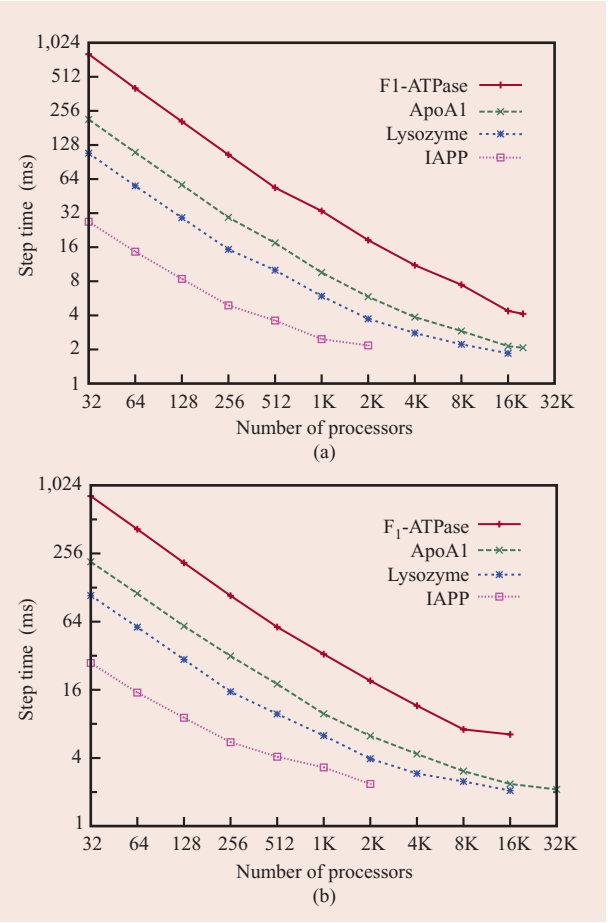

Figure 4

NAMD performance on various benchmarks: (a) coprocessor mode; (b) virtual node mode.

Table 2 Impact of performance optimizations for the ApoA1 benchmark on 4,096 processors applied in the given order. (The *XY* here represents the two-away optimization; i.e., two-away splitting along *X* and *Y* dimensions has been enabled.)

Version	*Time* (ms/step)
MPI	14.13
MPI with topology optimization	12.91
Native without progress calls	8.93
Native with progress calls	8.62
Native with two-away *XY*	3.48
Native with spanning tree	3.4
One-dimensional PME	4.7
Two-dimensional pencil PME	4.3

Table 2 presents the gains of the various performance optimizations described in the previous sections for the ApoA1 system on 4,096 processors, with the cutoff and PME computation. Because we wanted to isolate the gains of each of the optimizations, we disabled PME for the first six runs. We observed that the performance gained from optimizations often depends on the order in which they are applied. From Table 2, we can conclude that the two most effective optimizations correspond to the two-away communication and the Charm++ native machine layer. Even though not clearly reflected in the table, spanning trees are much more effective on larger processor partitions.

The scaling of NAMD with PME on four molecular systems is presented in **Figures 4(a)** and **4(b)**. The full electrostatic (PME) frequency for each of these runs was chosen on the basis of the timestep of the simulation, and this frequency was 2 for IAPP and lysozyme and 4 for ApoA1 and F_1-ATPase. All of these performance runs used the native layer of Charm++ with spanning trees and immediate messages enabled. The performance presented here excludes I/O overheads. The coprocessor mode results exhibit decreasing timesteps up to 16,384 CPUs for lysozyme and up to 20,480 CPUs for both ApoA1 and F_1-ATPase. The virtual node mode results for lysozyme and ApoA1 have decreasing timesteps up to 16,384 CPUs (8,192 nodes). **Table 3** shows the best performance and speedups achieved on the different benchmarks and the two-away options used for them. The speedup is computed from the NAMD performance on the smallest processor partition that has enough memory to run the benchmark. We have found that the two-away options have significant grain-size overheads, and this could be a reason for the limited scaling of NAMD. Note that for IAPP, lysozyme, and ApoA1, the performance saturates at about 2-ms. We are exploring new schemes to further improve the scaling of the NAMD application.

Related work

Blue Matter [10] is another MD application that has demonstrated very good performance on 16,384 nodes of the BG/L system. Blue Matter also uses a spatial decomposition algorithm, although this algorithm is different from the one used in NAMD, and Blue Matter uses low-level message-passing primitives. Blue Matter uses 2D decomposition for the PME computation and an optimized FFT library that scales to 16,384 nodes of the BG/L system [23].

So far, we have kept the NAMD software quite general. Architecture-specific optimizations are made available to NAMD through abstractions in the Charm++ runtime code. **Table 4** compares the performance of NAMD with that of Blue Matter. NAMD performance is better than that of Blue Matter at

Table 3 NAMD benchmark best timesteps on the BG/L system in coprocessor mode.

Benchmark	*Processors*	*Best timestep* (ms)	*Speedup*	*Options*
IAPP	2,048	2.17	315	Two-away *X, Y, Z*
Lysozyme	16,384	1.85	1,580	Two-away *X, Y, Z*
ApoAl	20,480	2.07	2,981	Two-away *X, Y, Z*
ATPase	20,480	4.13	5,918	Two-away *X, Y*

Table 4 Comparison of NAMD and Blue Matter: Times are in milliseconds. (CO: coprocessor; VN: virtual node.)

Number of nodes	512	1,024	2,048	4,096	8,192	16,384
Blue Matter (ms)	38.42	18.95	9.97	5.39	3.14	2.09
NAMD CO mode (ms)	18.6	9.56	5.84	3.86	2.91	2.14
NAMD VN mode (ms)	11.3	6.26	4.34	3.06	2.36	2.11

small processor partition sizes, but at the limits of scalability, its performance is similar to that of Blue Matter.

Remaining challenges

PME

As expected, our measurements confirm that the new pencil decomposition of the 3D FFT is significantly faster than the plane decomposition. The main bottleneck in PME now is the patch-to-pencil communication, which has relatively large messages compared with the transpose messages. We are exploring new mapping and decomposition schemes to optimize this data movement operation. We are also exploring new low-latency message-passing optimizations to further improve the performance of the PME 3D-FFT calculation.

Spanning trees

It is clear from our experiments that the spanning trees are useful for communicating coordinates from patches to compute objects and for collecting forces back from them. Without these trees, each patch will send approximately 60 to 80 messages and receive as many in each step. We use a two-level spanning tree with a branching factor of about 10. However, the spanning tree intermediate nodes (STINs) present a new challenge: The spanning tree can be determined only after the load balancer decides where to migrate *computes*, if any. However, when an STIN is placed on a processor, it may become overloaded. Even worse, because each patch creates a spanning tree for its clients independently, multiple STINs may be assigned to a processor. We used a centralized strategy to create all spanning trees together in order to reduce the number of STINs assigned to a processor, which helps improve performance. However, even one STIN adds a few hundred microseconds of overhead to a node (counting both a downward and an upward path through it). **Figure 5** is a projections timeline view of a 16,384-node run of NAMD with the ApoA1 benchmark. The color code for this projections plot is as follows. Red corresponds to the integrate computation, blue the force computation, and pink the spanning tree. Black regions represent communication overhead in the message layer, and white represents idle time.

This figure clearly shows the communication overhead of spanning trees on processor ID 15,236. Thus, more-powerful techniques are needed to break the circular dependence between STIN placement and load balancing. We plan to explore the simultaneous creation of STINs as a part of load balancing. Alternatively, it will be helpful to utilize a packet-level multicast strategy [as used by the SPI (system programming interface) layer in Blue Matter], possibly combined with either packet-level or higher-level reduction. Lower-level support for such overlapping multicasts, in which each processor sends data to 60 to 80 destinations, in future computers will be critical for continued performance improvements.

Summary and future work

We have described the basic parallelization strategies used by NAMD and how it was optimized for the BG/L supercomputer. Several new optimizations were necessary to tune the performance of NAMD on the BG/L system. Some of these optimizations were motivated by the available number of processors, which was an order of magnitude larger than the largest previous machine for NAMD. Other optimizations were motivated by the

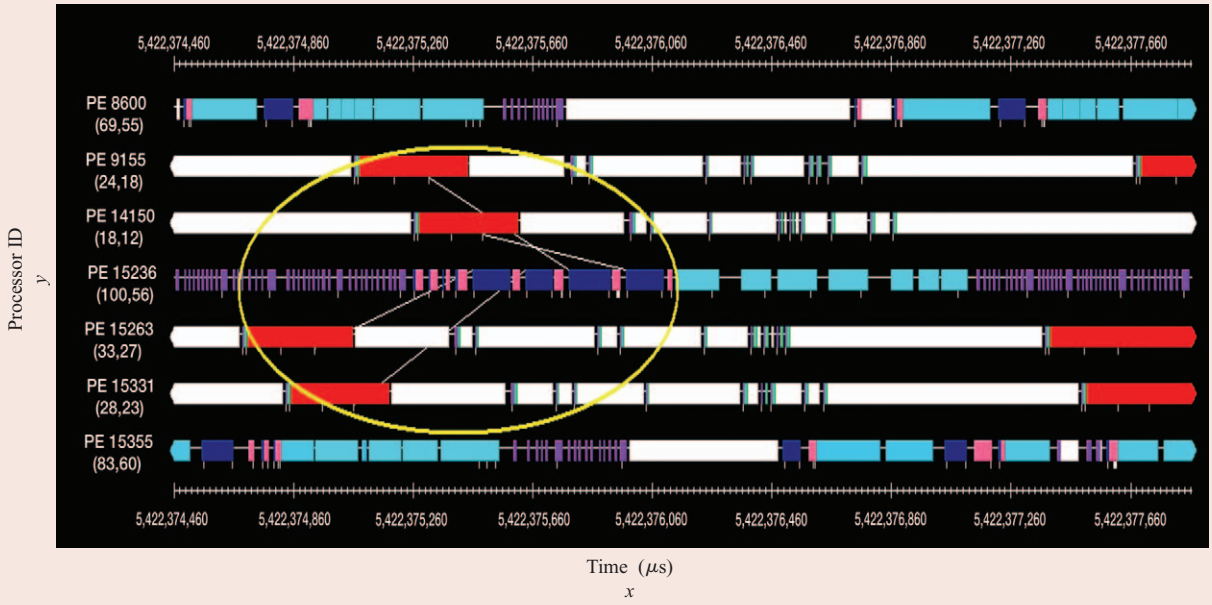

Figure 5

Projections snapshot showing performance bottlenecks. The seven horizontal bars display processor activity. The yellow oval highlights a region of interest, namely the overhead of the spanning tree. The four diagonal lines in the oval indicate communication, with the red bars as the sources of spanning trees.

various challenges of the BG/L architecture. We have presented an overview of these optimizations and performance data that shows that simulations of even a relatively small (92,224-atom) system perform quite well on 20,480 processors.

In addition to the immediate challenges identified in the previous section, the NAMD team is planning to incorporate techniques that reduce the memory footprint per processor, leading to simulations of larger molecular systems, as well as to parallelize the NAMD I/O. Optimizations for other machines, including the Cray XT3** and the forthcoming Blue Gene/P* system, are also planned.

Acknowledgments

We acknowledge the various students and staff of the Parallel Programming Laboratory and the Theoretical and Computational Biophysics Group at the University of Illinois, for their assistance in developing NAMD on the BG/L system. We thank Glenn Martyna and Fred Mintzer for time allocation on the BG/L machine located at the IBM T. J. Watson Research Center. We thank T. J. C. Ward, Mark Giampapa, Mark Mendell, and the compiler group at IBM Toronto for their help in optimizing the sequential performance of NAMD on PowerPC Architectures. We thank Gheorghe Almasi and Gabor Dozsa for assistance with the BG/L message layer. We also thank Philip Heidelberger for providing technical insights on the BG/L torus interconnection network. This work was supported by the National Institutes of Health (NIH PHS 5 P41 RR05969–04).

*Trademark, service mark, or registered trademark of International Business Machines Corporation in the United States, other countries, or both.

**Trademark, service mark, or registered trademark of Cray, Inc., in the United States, other countries, or both.

References

1. Y. Sugita and Y. Okamoto, "Replica-Exchange Molecular Dynamics Method for Protein Folding," *Chem. Phys. Lett.* **314**, No. 1, 141–151 (1999).
2. M. Sotomayor and K. Schulten, "Single-Molecule Experiments In Vitro and In Silico," *Science* **316**, No. 5828, 1144–1148 (2007).
3. J. C. Phillips, W. Wriggers, Z. Li, A. Jonas, and K. Schulten, "Predicting the Structure of Apolipoprotein A-I in Reconstituted High-Density Lipoprotein Disks," *Biophys J.* **73**, No. 5, 2337–2346 (1997).
4. L. Kalé, R. Skeel, M. Bhandarkar, R. Brunner, A. Gursoy, N. Krawetz, J. Phillips, A. Shinozaki, K. Varadarajan, and K. Schulten, "NAMD2: Greater Scalability for Parallel Molecular Dynamics," *J. Comput. Phys.* **151**, 283–312 (1999).
5. A. Y. Grama, A. Gupta, and V. Kumar, "Isoefficiency: Measuring the Scalability of Parallel Algorithms and

Architectures," *IEEE Parallel & Distributed Technol. Syst. & Technol.* **1**, No. 3, 12–21 (1993).
6. S. J. Plimpton and B. A. Hendrickson, "A New Parallel Method for Molecular-Dynamics Simulation of Macromolecular Systems," *J. Comput. Chem.* **17**, No. 3, 326–337 (1996).
7. Y.-S. Hwang, R. Das, J. H. Saltz, M. Hodoscek, and B. R. Brooks, "Parallelizing Molecular Dynamics Programs for Distributed-Memory Machines," *IEEE Comput. Sci. Eng.* **2**, No. 2, 18–29 (1995).
8. G. S. Almasi, C. Caşcaval, J. G. Castanos, M. Denneau, W. Donath, M. Eleftheriou, M. Giampapa, et al., "Demonstrating the Scalability of a Molecular Dynamics Application on a Petaflop Computer," *Proceedings of the 15th International Conference on Supercomputing*, New York, 2001, pp. 393–406.
9. K. J. Bowers, E. Chow, H. Xu, R. O. Dror, M. P. Eastwood, B. A. Gregersen, J. L. Klepeis, et al., "Molecular Dynamics—Scalable Algorithms for Molecular Dynamics Simulations on Commodity Clusters," *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, Tampa, FL, 2006, Article 84.
10. B. G. Fitch, A. Rayshubskiy, M. Eleftheriou, T. J. C. Ward, M. Giampapa, M. C. Pitman, and R. S. Germain, "Blue Matter: Approaching the Limits of Concurrency for Classical Molecular Dynamics," *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, Tampa, FL, 2006, Article 87.
11. S. Chatterjee, L. R. Bachega, P. Bergner, K. A. Dockser, J. A. Gunnels, M. Gupta, F. G. Gustavson, et al., "Design and Exploitation of a High-Performance SIMD Floating-Point Unit for Blue Gene/L," *IBM J. Res. & Dev.* **49**, No. 2/3, 377–391 (2005).
12. N. R. Adiga, M. A. Blumrich, D. Chen, P. Coteus, A. Gara, M. E. Giampapa, P. Heidelberger, et al., "Blue Gene/L Torus Interconnection Network," *IBM J. Res. & Dev.* **49**, No. 2/3, 265–276 (2005).
13. H. D. Simon, "Partitioning of Unstructured Problems for Parallel Processing," *Comput. Syst. Eng.* **2**, No. 2/3, 135–148 (1991).
14. M. Blocksome, C. Archer, T. Inglett, P. McCarthy, M. Mundy, J. Ratterman, A. Sidelnik, et al., "Design and Implementation of a One-Sided Communication Interface for the IBM eServer Blue Gene® Supercomputer," *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, Tampa, FL, 2006, Article No. 120.
15. G. Almasi, C. Archer, J. G. Castanos, J. A. Gunnels, C. C. Erway, P. Heidelberger, X. Martorell, et al., "Design and Implementation of Message-Passing Services for the Blue Gene/L Supercomputer," *IBM J. Res. & Dev.* **49**, No. 2/3, 393–406 (2005).
16. R. K. Brunner and L. V. Kalé, "Handling Application-Induced Load Imbalance Using Parallel Objects," *Parallel and Distributed Computing for Symbolic and Irregular Applications*, World Scientific Publishing, 2000, pp. 167–181.
17. G. Zheng, "Achieving High Performance on Extremely Large Parallel Machines," Ph.D. dissertation, University of Illinois at Urbana-Champaign, 2005.
18. L. V. Kalé, "The Virtualization Model of Parallel Programming: Runtime Optimizations and the State of Art," *LACSI 2002*, Albuquerque, NM, October 2002.
19. S. Kumar, C. Huang, G. Almasi, and L. V. Kalé, "Achieving Strong Scaling with NAMD on Blue Gene/L," *20th International Parallel and Distributed Processing Symposium*, April 2006.
20. T. Darden, D. York, and L. Pedersen, "Particle mesh Ewald. An $N\cdot\log(N)$ Method for Ewald Sums in Large Systems," *J. Chem. Phys.* **98**, 10089–10092 (1993).
21. P. Westermark, U. Engstrom, K. H. Johnson, G. T. Westermark, and C. Betsholtz, "Islet Amyloid Polypeptide: Pinpointing Amino Acid Residues Linked to Amyloid Fibril Formation," *Proc. Natl. Acad. Sci.* **87**, 5036–5040 (1990).
22. R. Zhou, M. Eleftheriou, A. K. Royyuru, and B. J. Berne, "Destruction of Long-Range Interactions by a Single Mutation in Lysozyme," *Proc. Natl. Acad. Sci.* **104**, No. 14, 5824–5829 (2007).
23. M. Eleftheriou, B. Fitch, A. Rayshubskiy, T. J. C. Ward, and R. Germain, "Performance Measurements of the 3D FFT on the Blue Gene/L Supercomputer," *Proceedings of the 11th International Euro-Par Conference*, 2005, pp. 270–803.

Received March 16, 2007; accepted for publication April 11, 2007; Internet publication January 16, 2008

Sameer Kumar *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (sameerk@us.ibm.com)*. Dr. Kumar received a B.Tech. (1999) degree in computer science from Indian Institute of Technology, Madras, India, and an M.S. and Ph.D. degrees in computer science from the University of Illinois at Urbana-Champaign. His Ph.D. thesis focused on optimizing communication for massively parallel processing. He is currently a Research Staff Member at the T. J. Watson Research Center and is working on the Blue Gene* Project. His research interests include scaling parallel applications to massively parallel machines and next-generation interconnection network design. He coauthored a paper on scaling the molecular dynamics program NAMD, and it was one of the winners of the Gordon Bell Prize at the 2002 Supercomputing Conference.

Chao Huang *Department of Computer Science, Thomas M. Siebel Center, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801 (chuang10@uiuc.edu)*. Mr. Huang received a B.E. degree in computer science from Tsinghua University, Beijing, in 2001 and an M.S. degree in computer science from the University of Illinois at Urbana-Champaign in 2004. Mr. Huang is a Ph.D. candidate at the Parallel Programming Laboratory at the University of Illinois. His research focuses on higher-level language that allows expression of overall flow of control in complicated parallel programs.

Gengbin Zheng *Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801 (gzheng@uiuc.edu)*. Dr. Zheng received B.S. (1995) and M.S. (1998) degrees in computer science from the Beijing University, Beijing, China. His master's thesis concerned a high-performance Fortran compiler. He received a Ph.D. degree in computer science from the University of Illinois at Urbana-Champaign in 2005. He is a postdoctoral research associate in computer science and engineering, working with both the Center for Simulation of Advanced Rockets and Professor Laxmikant V. Kalé. His research interests span various aspects of parallel computing, including dynamic automatic load balancing to scale highly adaptive parallel applications to a large number of processors, simulation-based method to predict performance of applications for large parallel machines, and fault tolerance. He coauthored a paper on scaling the molecular dynamics program NAMD, and it was one of the winners of the Gordon Bell Prize at the 2002 Supercomputing Conference.

Eric Bohm *Department of Computer Science, Thomas M. Siebel Center, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801 (ebohm@uiuc.edu)*. Mr. Bohm received a B.S. degree in computer science from the State University of New York at Buffalo in 1992. He worked as Director of Software Development for Latpon Corporation from 1992 to 1995, and next as Director of National Software Development from 1995 to 1996. He worked as Enterprise Application Architect at MEDE America from 1996 to 1999 and as an Application Architect at WebMD from 1999 to 2001. Following a career shift toward academia, he joined the Parallel Programming Lab at University of Illinois at Urbana-Champaign in 2003. His current focus as a Research Programmer is on optimizing molecular dynamics codes for tens of thousands of processors.

Abhinav Bhatele *Department of Computer Science, Thomas M. Siebel Center, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801 (bhatele2@uiuc.edu)*. Mr. Bhatele received a B.Tech. degree in computer science and engineering from Indian Institute of Technology, Kanpur, India, in 2005. He is a Ph.D. student at the Parallel Programming Lab at the University of Illinois. His research is centered on topology-aware mapping and load balancing for parallel applications. He is a co-developer of the molecular dynamics applications NAMD and LeanCP, developed at the Parallel Programming Lab.

James C. Phillips *Theoretical and Computational Bio-Physics Group, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801*. Dr. Phillips received his B.S. degree in physics and mathematics from Marquette University, Milwaukee, Wisconsin, in 1993, and his M.S. and Ph.D. degrees in physics from the University of Illinois at Urbana-Champaign in 1994 and 2002, respectively. He was supported by a Fannie and John Hertz Graduate Fellowship and a Department of Energy Computational Science Graduate Fellowship. He is currently a Senior Research Programmer in the Theoretical and Computational Biophysics Group at the Beckman Institute for Advanced Science and Technology, University of Illinois at Urbana-Champaign. Dr. Phillips is the lead developer of the scalable molecular dynamics program NAMD, which received a 2002 Gordon Bell Prize, and the NAMD application has been cited in more than 500 published technical papers. His research interests span the field of biomolecular simulation, from potential functions and simulation protocols to numerical methods and parallel computing.

Hao Yu *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (yuh@us.ibm.com)*. Dr. Yu received B.S. and M.S. degrees in computer science from Tsinghua University, Beijing, China, in 1994 and 1997. He received his Ph.D. degree in computer science from Texas A&M University, College Station, Texas, in 2004. He is currently a postdoctoral researcher at the IBM T. J. Watson Research Center. His research interests include compiler optimization for high-performance computing, system software for scalable systems, and parallel I/O.

Laxmikant V. Kalé *Department of Computer Science, Thomas M. Siebel Center, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801 (kale@uiuc.edu)*. Professor Kalé has been working on various aspects of parallel computing, with a focus on enhancing performance and productivity via adaptive runtime systems. His collaborations involve the widely used biomolecular simulation program NAMD, which won the Gordon Bell Prize at the Supercomputing Conference in 2002. Other collaborative work focuses on computational cosmology, quantum chemistry, rocket simulation, space-time meshes, and other unstructured mesh applications. His group successfully distributes and supports software embodying his research ideas involving Charm++, adaptive MPI, and the ParFUM framework. Professor Kalé received the B.Tech. degree in electronics engineering from Benares Hindu University, Varanasi, India, in 1977, and an M.E. degree in computer science from Indian Institute of Science in Bangalore, India, in 1979. He received a Ph.D. degree in computer science from the State University of New York, Stony Brook, in 1985. He worked as a scientist at the Tata Institute of Fundamental Research from 1979 to 1981. He joined the faculty of the University of Illinois at Urbana-Champaign as an Assistant Professor in 1985, where he is currently employed as a professor.