Scalable Molecular Dynamics with NAMD on Blue Gene/L

Sameer Kumar[1], Chao Huang[2], Gengbin Zheng[2], Eric Bohm[2]
Abhinav Bhatele[2], James C. Phillips[3], Hao Yu[1]
Laxmikant V. Kalé[2]

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

[2] Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA
{chuang10, gzheng, ebohm, bhatele2, kale}@uiuc.edu

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

Abstract

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 need to be executed to understand interesting biological phenomenon. Hence we need MD applications to scale to thousands of processors, even though the individual time step 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 get good sequential performance with NAMD on the embedded 440 core. We also present several techniques to scale NAMD to 20,480 nodes of Blue Gene/L. These include topology specific optimizations to localize communication, new messaging protocols that are optimized for the Blue Gene/L torus (as they do not require message ordering), topology aware load balancing, and overlap of computation and communication. We also present performance results of various molecular systems with sizes ranging from 5570 to 327,506 atoms.

Introduction

With a greater understanding of the functioning of biological systems, the importance of biomolecular simulations has increased significantly. Over 43,000 structures are publicly available from the Protein Data Bank (www.pdb.org). This has enabled researchers to explore the relationship between structure and function through simulations that are now based on a firm experimental foundation. For the majority of proteins, only amino-acid sequences are known and molecular simulations are needed to predict detailed 3-D structure which is the famously difficult "protein folding problem."

Such molecular simulations present several computational challenges. Since hydrogen atoms in a biomolecule vibrate with a period of approximately 10 femtoseconds, the time step needs to be around 1 femtosecond for stable and accurate integration. Yet, the phenomenon of interest may occur only at a scale of microseconds. Several nanoseconds of simulation may be needed even to allow a protein to relax into the nearest low-energy state. Some phenomena can be studied by observing the molecule's behavior over tens of nanoseconds. We can set the simulation up in an initial state and force it through interesting behavior paths. In spite of these shortcuts, we are left with the problem of simulating several million to a few billion time steps.

Further, the number of atoms in a particular biologically interesting configuration is fixed. For example, if we want to study a particular Aquaporin, which straddles a cell membrane and allows water to cross the membrane, we need a simulation involving one Aquaporin tetramer, a reasonably-sized patch of cell membrane in which to embed it, and sufficient quantity of water molecules around the structure. This may comprise a few hundred thousand atoms. Just because we have a larger computer available, we cannot increase the **size** (i.e. the number of atoms involved) of this simulation, because the molecule/s being studied involves a fixed number of atoms. This is unlike the study of continuous phenomena, such as weather prediction, where we can simply increase the resolution to exploit a large machine. Of course, we can study larger and larger molecules as larger machines become available, but (a) the size of such molecular assemblies of interest is normally limited (b) we still need to study a particular system to understand its behavior, and the real challenge is to simulate a large number of time steps fast.

For some problems, multiple independent or weakly coupled (as in replica-exchange methods [1]) simulations may be used to increase sampling of molecular configurations. For many simulations, however, artificial steering forces [2] are used to drive the molecule through a series of transitions (such as the rotary mechanism of ATP synthase). Larger steering forces contribute artifacts, so for accuracy, simulations are done using the smallest steering forces that provide results in a reasonable time. Hence, 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 performance than implied by their atom counts, and the challenge of reducing wallclock time per step remains.

As a concrete goal, consider the Apolipoprotein-A1 (ApoA1) system, a model of a newly-formed high-density lipoprotein (HDL) particle [3]. (HDL transports cholesterol in the bloodstream and is the "good" cholesterol measured by blood tests.) With 92,224 atoms and a mix of proteins, lipids, and water, ApoA1 is representative of modern moderately-sized simulations. Sequentially, the simulation of ApoA1 takes about 6.2 seconds per time step on a single processor core of BG/L. The challenge is to take the 6.2 seconds computation and run it efficiently on several thousand processors, with each time step taking only a few milliseconds.

In this paper, we describe how this scaling was accomplished in NAMD, specifically on the Blue Gene/L machine. NAMD [4] is a parallel molecular dynamics code developed at the University of Illinois at Urbana-Champaign, as a collaborative project involving the Theoretical and Computational Biophysics Group (http://www.ks.uiuc.edu) and the Computer Science Department, including the Parallel Programming Laboratory (http://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. This hybrid parallelization strategy has remained effective over the past ten years. We begin by describing and reviewing this strategy. We then present a series of optimizations, including sequential optimizations targeted towards IBM PowerPC 440 cores used in Blue Gene/L, as well as parallel optimizations including dynamic load balancing. We present performance data for molecular systems ranging from 5570 to 327,506 atoms.

NAMD Parallelization Strategy

NAMD uses a hybrid strategy that combines spatial decomposition with force decomposition, and couples it with Charm++'s dynamic load balancing framework. The dominant computation in molecular dynamics is that of computing non-bonded forces i.e. electrostatic and Van der Waal's 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 cut-off radius r_c, and separation of computation of short-range and long range forces. For each atom, the non-bonded forces due to atoms within r_c are calculated explicitly. The long-range forces due to the atoms outside this radius are calculated using a O($n\ log(n)$) Particle Mesh Ewald (PME) algorithm. Even with this splitting, 90% of the computation cost is due to explicit calculation of non-bonded forces within the cut-off radius.

Early attempts at parallel molecular dynamics (in 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 number of processors, and in a way that does not even allow "weak scaling" – i.e. even if we were to increase the number of atoms, the communication-to-computation ratio does not improve. A pure force decomposition scheme, such as that in [6, 7], also suffers from this limit.

We also showed in [4] that spatial decomposition does not suffer from 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 cut-off distance r_c. Let $B = r_c + r_H + m$, where r_H is twice the maximum length of a bond to a H atom and m, the margin is twice the distance which atoms may move without migration between patches being necessary. Then b along each dimension is chosen as B/k. Typically, k is either one or two (although we have experimented with $k = 3$). With $k = 1$ (also called 1-away decomposition), there are about 400-700 atoms per cube. With $k = 2$, this drops down to 50-75 atoms. To provide intermediate granularities, we support non-cubic patches: each dimension can be either B or $B/2$, for example. With $k = 1$, only atoms in neighboring patches need to interact (i.e. there are 27 interactions each patch participates in). With $k = 2$, interactions involve 125 cubes that are “2-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 every pair of interacting patches, NAMD creates a force-computation-object (just 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.

More recent proposals for scalable molecular dynamics [8, 9, 10] use the basic strategy of hybrid decomposition, which originated from NAMD 2.0. They differ in how the force computations are assigned to processors: either via static (but topology sensitive) schemes [8, 9] or the Blue Matter scheme [10] that allocates work based on number of atoms. We believe that the original scheme is at least as good or better than these schemes because of its ability to take the dynamic processor load into account (assuming 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 programmer’s ontology 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 it is destined for, 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. Based on these measurements, the load balancer can reassign objects accurately to improve load balance and to decrease communication.

Blue Gene/L Optimizations

Improving Sequential Performance on the 440 Core

The Blue Gene/L machine is based on the embedded PowerPC 440 core. It took a significant effort to achieve good sequential performance on the Blue Gene/L embedded core. Due to aliasing constraints in the NAMD inner loop, the IBM XL 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 get the best performance. We observed that the bonded code in NAMD had several stack temporaries as it was using C++ operator overloading. The stack temporaries were introducing expensive loads, stores and pipeline stalls. We reported this to the IBM XL compiler team and obtained a fix which is now available in XLC Version 8.0. The net result of the above optimizations was to more than double serial performance on the PowerPC 440. The optimizations also helped other PowerPC architectures. Table **??** compares the performance of NAMD version 2.5 with NAMD version 2.7 pre-release. (The Blue Gene/L optimizations in NAMD were added after the release of NAMD version 2.5.)

The PowerPC 440 core is enhanced with an additional floating point unit called the double floating point unit (double FPU) [11]. To take advantage of the double FPU, the addresses of loads and stores have to 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 X, Y and Z dimensions to be computed. However, the SIMD instructions can only parallelize the X and Y dimensions and the computation of the Z dimension is not SIMDized. 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 **??**. We are working on further optimizing the SIMD version of NAMD. A possible reason for lower 440d performance could be a cache miss in the force compute loop. This loop uses interpolation tables for the three independent terms in the potential (electrostatics plus Lennard-Jones r^{-12} and r^{-6}), thus eliminating *reciprocal square root* and *erfc* computations. The interpolation table is quite large and of the order of several hundred cache lines. Many PowerPC architectures have relatively small L1 caches (32 KB on Blue Gene/L). This may lead to cache misses in the inner compute loop of NAMD. As the access pattern to the interpolation table is quite irregular, even the L2 prefetch unit on the Blue Gene/L chip may not be very effective in this case.

We are exploring new computational algorithms which nay 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 to further reduce the number of entries in the interpolation table. We hope these optimizations will further improve the performance of NAMD on Blue Gene/L and other PowerPC architectures.

Topology Mapping

The Blue Gene/L supercomputer has a torus interconnection network [12] for application data exchange. As 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 Blue Gene/L. The patches in NAMD are allocated to processors using an orthogonal recursive bisection (ORB) scheme [13] to map the patch objects to the Blue Gene/L torus. The dimensions of the Blue Gene/L torus depend on the size of the processor partition, while the dimensions of the patch torus depend on the problem's size. Both the processor and patch tori are typically not cubic. So, 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 patch torus. For example, when we map a $13 \times 6 \times 4$ patch torus to a $8 \times 32 \times 16$ processor torus we get the following axis-map $X_{procesor} = Z_{patch}, Y_{processor} = X_{patch}, Z_{processor} = Y_{patch}$. 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 into two sub-grids, where the size of each partition corresponds to the load of the patch partition. This is repeated recursively till we have one patch, which is allocated to a random processor in the corresponding processor sub-grid. Figure **??** shows the mapping of patches and computes onto the 3D torus of Blue Gene/L.

Communication Optimizations

We have developed a *native* Charm++ runtime-system optimized for Blue Gene/L on top of the Blue Gene message layer [14]. We found the Charm++ MPI driver unsuitable as it called MPI_Iprobe and MPI_Test to make progress on the network, thus introducing overheads. An optimized runtime also allowed us to explore the adaptive-eager messaging protocol that does not require the ordering semantics of MPI.

Adaptive Eager Protocol

The production MPI software [15] on Blue Gene/L has two protocols for point-to-point messages, (i) Eager and (ii) 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 receiver is ready to receive it sends a CTS (clear-to-send) packet back to the sender. On receiving the CTS, the sender sends the application buffer with adaptive routing.

As the eager protocol uses deterministic routing it can restrict the application's communication performance. 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 in size.

We present the adaptive eager protocol to optimize the scenario where the application sends several short messages a few kilobytes in size. 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 has to carry all the state for the message. In MPI, this state would include the communicator and tag for the message along with the source and the size. So, for an MPI implementation of 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 adaptive eager protocol in MPI will be lower than that for Eager or Rendezvous. Moreover, only one message can be outstanding as messages need to be matched in order.

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

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, which is 16 outstanding messages with a 4 bit sequence number. Once the receiver has received all the 16 messages, it sends an acknowledgment back to the sender to send the next set of 16 messages.

Dynamic Load-Balancing

NAMD uses the Charm++ dynamic load balancing framework [16, 17]. The patches have an initial placement in a topology optimized manner using the ORB scheme presented in the Topology Mapping subsection. To achieve good load balance, it is essential for the Charm++ runtime to record the most up-to-date application and system load information. The Charm++ runtime exploits a simple heuristic called *Principle of Persistence* [18] to obtain load information automatically. 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 molecular dynamics simulations where 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 of 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 non-migratable work on the processors. The load statistics are provided as a parameter to a load balancing strategy which computes the new object placements.

We use two load balancing strategies in NAMD. The first scheme is a comprehensive load balancing strategy which assigns migratable work *from scratch* (mostly non-bonded force computation), ignoring the current location of such work. This comprehensive scheme is performed only once during a run of NAMD. The second load balancing strategy is a refinement strategy that just moves a few objects from overloaded processors to lightly loaded processors. The refinement scheme is called periodically (every few thousand time steps) to move compute objects to balance changes in processor load for atom migrations.

Both the comprehensive and refinement strategies have topology optimizations built into them. We present each of them below:

- Comprehensive Strategy: In this strategy, the load balancer first assigns all the compute objects to a max-heap. The strategy then picks the heaviest compute object and assigns it to the processor based on a greedy heuristic which takes into account the processor load, communication history and the processor's nearness on the Blue Gene/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 which are close to the patch processor on the Blue Gene/L torus. It also prefers processors which are less than 4 hops from the midpoint of the patches whose interaction is being computed in the compute object.

- Refinement Strategy: In this strategy, the overloaded processors are first allocated to a max heap. The strategy goes in a loop picking the highest loaded processors, and then removing the heavy objects in that processor to a lightly loaded accepting processor. This accepting processor is chosen using heuristics similar to those in the comprehensive scheme. Refinement strategy iterates till there are no overloaded processors above a threshold load. The refinement strategy prefers lightly loaded processors within eight hops of the mid-point of the patches whose interaction is being computed.

Once the load balancing strategy has finished re-assigning compute objects, the Charm++ runtime moves the objects to their new destinations. After this is 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 **??** shows patches and their multicast targets superimposed on a 2D view of the physical processor topology. A two-level k-ary tree is generally used in NAMD where k is close to ten.

Overlap of Computation and Communication

The Blue Gene/L machine has two PowerPC 440 cores on each node. However, it does not have a DMA unit on the compute node. Ideally one of the cores could serve as a communication co-processor, but due to lack of cache-coherence the caches have to be flushed for any communication between the cores. The overhead of cache flushing may limit the performance of co-processor mode as 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 6 normal priority Torus FIFOs, and each of these FIFOs can store up to 4 packets. At full link bandwidth of 175 MB/s each FIFO would fill up in about 4320 processor cycles. We have observed in NAMD that the achievable throughput due to network contention is only about 2 links, which implies that each FIFO would fill up every 12960 cycles on average. We can make the cores compute for these 12960 cycles, and periodically make calls to drain network FIFOs and call the progress engine in the messaging software. In NAMD, the rate of progress is a command line parameter and can be tuned to the processor partition size and the benchmark.

The Blue Gene/L torus interconnect is a reliable network where a packet is only sent downstream 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 need to be drained before they fill up. If this is not done, packets are stuck 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 to support such progress calls from within 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 (Dynamic Load Balancing Subsection). If an intermediate destination on the spanning-tree is busy in a compute loop, the multicast messages will be delayed resulting in bad 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 which are sent back to the patches.

Particle Mesh Ewald

NAMD uses the Particle Mesh Ewald method [20] to compute the long-range interactions between the atoms. PME requires two 3D Fast Fourier Transforms to be computed. NAMD 2.6 used a 1D decomposition for the FFT operations. Since the 1D decomposition only requires 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, 108 processors for the ApoA1 benchmark. However, the message-driven execution model of Charm++ allows the 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. Still, we have observed that this 1D decomposition did not scale on Blue Gene/L and many other architectures due to insufficient parallelism.

We implemented a 2D decomposition for PME, where the FFT calculation is decomposed into thick-pencils with 3 phases of computation and 2 phases of transpose communication. A *thick-pencil* along the X dimension will have all the FFT grid points in the X dimension, while the Y and Z dimension sizes would typically vary from 1 to 4. The FFT operation is computed by 3 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 two.

In addition to the 2 FFT calculations, PME in NAMD has 2 additional 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 4x4x4. 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 do a forward 3D FFT followed by the Ewald calculation on the transformed grid in k-space. Next, a backward 3D FFT is performed which computes the long-range forces which 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 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. Consider a typical situation where 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 benchmark the performance of NAMD on Blue Gene/L. These are the 5570 atom Islet Amyloid Polypeptide system (IAPP) [21], the Lysozyme in urea simulation [22] (39,864 atoms), Apolipoprotein-A1 (92,224 atoms) and the F1-ATPase system (327,506) atoms.

Table **??** presents the gains of the various performance optimizations presented in the previous sections for the ApoA1 system on 4096 processors doing cutoff and PME computation. As we wanted to isolate the gains of each of the optimizations we disabled PME for the first six runs. We have observed that the performance gained from optimizations often depends on the order in which they are applied. From Table **??**, we can conclude that the two most effective optimizations are 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 **??** and **??**. The full electrostatics (PME) frequency for each of these runs was chosen based on the time-step of the simulation. It was 2 for IAPP and Lysozyme, while it was 4 for ApoA1 and F1-ATPase. All 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 co-processor mode results have decreasing time steps to 16384 CPUs for Lysozyme, and to 20,480 CPUs for both ApoA1 and F1-ATPase. The virtual node mode results for Lysozyme and ApoA1 have decreasing time steps up till 16,384 CPUs (8192 nodes). Table **??** shows the best performance and speedups achieved on the different benchmarks and the two-away options used for them. The speedup is computed from NAMD performance on the smallest processor partition which has enough memory to run the benchmark. We have found that the two-away options have significant grain size overheads, and that could be a reason for the limited scaling of NAMD. Observe that for IAPP, Lysozyme and APoA1 the performance saturates at about the 2ms mark. We are exploring new schemes to further improve the scaling of the NAMD application.

Related Work

Blue Matter [10] is another application which has demonstrated very good performance on 16,384 nodes of Blue Gene/L. Blue Matter also uses a spatial decomposition algorithm, though different from the one used in NAMD. Besides, it uses low level message passing primitives. Blue Matter has 2D decomposition for the PME computation and uses an optimized FFT library which scales to 16384 nodes of Blue Gene/L [23].

So far we have kept the NAMD software quite general. Architecture specific optimizations are made available to NAMD through abstractions in Charm++ runtime. Table **??** compares the performance of NAMD with Blue Matter. NAMD performance is better than Blue Matter at small processor partition sizes, but at the limits of scalability its performance is similar to Blue Matter.

Remaining Challenges

PME: As expected, our measurements confirm that the new pencil decomposition of 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 as compared to 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 3-D 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 them, each patch will send around 60 to 80 messages, and receive as many, in each step. We use a 2-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 decided 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. What is worse, since each patch is creating 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 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 the downward and upward path through it). Figure **??** 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 is integrate computation, blue is force computation, pink is spanning tree, black regions is communication overhead in the message layer, and white represents idle time.

The figure clearly shows the overhead of spanning trees on processor 15236. So, stronger techniques are needed to break the circular dependence between STIN placement and load balancing. We plan to explore is 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 SPI layer in Blue Matter), possibly combined with either packet level or higher level reduction. Lower-level support for such overlapping multicasts (here: 60-80 destinations each, and often a 10th of all processors originating a separate multicast almost simultaneously) in future machines will be critical for continued performance improvements.

Summary and Future Work

We described the basic parallelization strategies used by NAMD, and how it was optimized for the Blue Gene/L supercomputer. Several new optimizations were necessary to tune performance of NAMD on Blue Gene/L. Some of these optimizations were motivated by the order of magnitude larger number of processors compared with the largest previous machine for NAMD, others by the various challenges of the Blue Gene/L architecture. We presented an overview of these optimizations, and presented 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, and to parallelize its I/O. Optimizations to other machines, including Cray XT3, and the upcoming Blue Gene/P are also planned.

Acknowledgments

We would like to 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 Blue Gene/L. We would like to thank Glenn Martyna and Fred Mintzer for time allocation on the BGW machine at the IBM T. J. Watson Research Center. We would like to thank T. J. C. Ward, Mark Giampapa and Mark Mendell and the Tobey group at IBM Toronto for their help in optimizing the sequential performance of NAMD on PowerPC architectures. We would like to thank Gheorghe Almasi and Gabor Dozsa for assistance with the Blue Gene/L message layer. We would also like to thank Philip Heidelberger for providing technical insights on the Blue Gene/L torus interconnection network. This work was supported by the National Institutes of Health (NIH PHS 5 P41 RR05969-04).

References

[1] Yuji Sugita and Yuko Okamoto. Replica-exchange molecular dynamics method for protein folding. 314:141–151, 1999.

[2] Marcos Sotomayor and Klaus Schulten. Single-molecule experiments in vitro and in silico. 316:1144–1148, 2007.

[3] James C. Phillips, Willy Wriggers, Zhigang Li, Ana Jonas, and Klaus Schulten. Predicting the structure of apolipoprotein A-I in reconstituted high density lipoprotein disks. 73:2337–2346, 1997.

[4] Laxmikant Kalé, Robert Skeel, Milind Bhandarkar, Robert Brunner, Attila Gursoy, Neal Krawetz, James Phillips, Aritomo Shinozaki, Krishnan Varadarajan, and Klaus Schulten. NAMD2: Greater scalability for parallel molecular dynamics. *Journal of Computational Physics*, 151:283–312, 1999.

[5] A. Grama, A. Gupta, and V. Kumar. Isoefficiency: Measuring the Scalability of Parallel Algorithms and Architectures. *IEEE Parallel & Distributed Technology*, 1(3), August 1993.

[6] 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.

[7] Y.-S. Hwang, R. Das, J.H. Saltz, M. Hodoscek, and B.R. Brooks. Parallelizing Molecular Dynamics Programs for Distributed Memory Machines. *IEEE Computational Science & Engineering*, 2(2):18–29, Summer 1995.

[8] George S. Almasi, Calin Cascaval, Jose G. Castanos, Monty Denneau, Wilm E. Donath, Maria Eleftheriou, Mark Giampapa, Howard Ho, Derek Lieber, Jose E. Moreira, Dennis M. Newns, Marc Snir, and Henry S. Warren Jr. Demonstrating the scalability of a molecular dynamics application on a petaflop computer. In *ICS '01: Proceedings of the 15th international conference on Supercomputing*, pages 393–406, New York, NY, USA, 2001. ACM Press.

[9] Kevin J. Bowers, Edmond Chow, Huafeng Xu, Ron O. Dror, Michael P. Eastwood, Brent A. Gregersen, John L. Klepeis, Istvan Kolossvary, Mark A. Moraes, Federico D. Sacerdoti, John K. Salmon, Yibing Shan, and David E. Shaw. Molecular dynamics—scalable algorithms for molecular dynamics simulations on commodity clusters. In *SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 84, New York, NY, USA, 2006. ACM Press.

[10] Blake G. Fitch, Aleksandr Rayshubskiy, Maria Eleftheriou, T. J. Christopher Ward, Mark Giampapa, Michael C. Pitman, and Robert S. Germain. Molecular dynamics—blue matter: approaching the limits of concurrency for classical molecular dynamics. In *SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 87, New York, NY, USA, 2006. ACM Press.

[11] S. Chatterjee, L. R. Bachega, P. Bergner, K. A. Dockser, J. A. Gunnels, M. Gupta, F. G. Gustavson, C. A. Lapkowski, G. K. Liu, M. Mendell, R. Nair, C. D. Wait, T. J. C. Ward, and P. Wu. Design and exploitation of a high-performance SIMD floating-point unit for Blue Gene/L. *IBM J. Res. Dev.*, 49(2/3):377–391, 2005.

[12] N. R. Adiga, M. A. Blumrich, D. Chen, P. Coteus, A. Gara, M. E. Giampapa, P. Heidelberger, S. Singh, B. D. Steinmacher-Burow, T. Takken, M. Tsao, and P. Vranas. Blue Gene/L torus interconnection network. *IBM J. Res. Dev.*, 49(2/3):265–276, 2005.

[13] H.D. Simon. Partitioning of Unstructured Problems for Parallel Processing. *Computing Systems in Engineering*, 2(2/3):135–148, 1991.

[14] Michael Blocksome, Charles Archer, Todd Inglett, Pat McCarthy, Mike Mundy, Joe Ratterman, Albert Sidelnik, Brian Smith, Gheorghe Almasi, Jose Castanos, Derek Lieber, Jose Moreira, Sriram Krishnamoorthy, and Vinod Tipparaju. Design and Implementation of One-Sided Communication for the IBM eServer Blue Gene Supercomputer. In *Proceedings of Supercomputing*, November 2006.

[15] G. Almasi, C. Archer, J. G. Castanos, J. A. Gunnels, C. C. Erway, P. Heidelberger, X. Martorell, J. E. Moreira, K. Pinnow, J. Ratterman, B. D. Steinmacher-Burow, W. Gropp, and B. Toonen. Design and implementation of message-passing services for the Blue Gene/L supercomputer. *IBM J. Res. Dev.*, 49(2/3):393–406, 2005.

[16] Robert K. Brunner and Laxmikant V. Kalé. Handling application-induced load imbalance using parallel objects. In *Parallel and Distributed Computing for Symbolic and Irregular Applications*, pages 167–181. World Scientific Publishing, 2000.

[17] Gengbin Zheng. *Achieving high performance on extremely large parallel machines: performance prediction and load balancing*. PhD thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, 2005.

[18] Laxmikant V. Kalé. The virtualization model of parallel programming : Runtime optimizations and the state of art. In *LACSI 2002*, Albuquerque, October 2002.

[19] Sameer Kumar, Chao Huang, Gheorghe Almasi, and Laxmikant V. Kalé. Achieving strong scaling with NAMD on Blue Gene/L. In *Proceedings of IEEE International Parallel and Distributed Processing Symposium 2006*, April 2006.

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

[21] Engstrom U. Westermark P., Johnson K.H., Westermark G.T., and Betsholtz C. Islet amyloid polypeptide: Pinpointing amino acid residues linked to amyloid fibril formation. *Proc. of National Academy of Sciences*, 87:5036–5040, 1990.

[22] Ruhong Zhou, Maria Eleftheriou, Ajay Royyuru, and Berne B. J. *Proc. of National Academy of Sciences (In Press)*, 104, 2007.

[23] M. Eleftheriou, B. G. Fitch, A. Rayshubskiy, T. J. C. Ward, and R. S. Germain. Performance Measurements of the 3d FFT on the Blue Gene/L Supercomputer. In *Proceedings of the 11th International Euro-Par Conference*, September 2005.

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 the B.Tech. (1999) degree in Computer Science from Indian Institute of Technology Madras, India and his MS and PhD degrees in Computer Science from the University of Illinois at Urbana Champaign. His Ph.D. thesis was 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. A paper co-authored by him on scaling the molecular dynamics program NAMD was one of the winners of the Gordon Bell award in SC2002.

Chao Huang Department of Computer Science, Thomas M. Siebel Center, University of Illinois at Urbana-Champaign, Urbana, IL 61801. (chuang10@uiuc.edu). Chao 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 is focused 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, IL 61801 (gzheng@uiuc.edu). Dr. Zheng received the BS (1995) and MS (1998) degrees in Computer Science from the Beijing University, Beijing, China. His Master's thesis was on High Performance Fortran compiler. He received a Ph.D. in Computer Science from University of Illinois at Urbana-Champaign in 2005. He is a Post-doctoral research associate at Computer Science and Engineering, working with both the Center for Simulation of Advanced Rockets and Prof. Kale. His research interests spans 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. A paper co-authored by him on scaling the molecular dynamics program NAMD was one of the winners of the Gordon Bell award in SC2002.

Eric Bohm Department of Computer Science, Thomas M. Siebel Center, University of Illinois at Urbana-Champaign, Urbana, IL 61801. (ebohm@uiuc.edu). Eric 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 Corp from 1992 to 1995, then as Director of National Software Development from 1995 to 1996. He worked as Enterprise Application Architect at MEDE America from 1996 to 1999. He completed his time in industry as Application Architect at WebMD from 1999 to 2001. Following a career shift towards 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, IL 61801. (bhatele2@uiuc.edu). Abhinav received a B. Tech. degree in Computer Science and Engineering from Indian Institute of Technology, Kanpur (INDIA) in May 2005. He is a Ph.D. student at the Parallel Programming Lab (PPL) at the University of Illinois. His research is centered around topology aware mapping and load balancing for parallel applications. He is a co-developer of the molecular dynamics applications developed at PPL, NAMD and LeanCP.

James C. Phillips Theoretical and Computational Bio-Physics Group, University of Illinois at Urbana-Champaign, Urbana IL, 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, Beckman Institute for Advanced Science and Technology, University of Illinois at Urbana-Champaign, the group where he did his thesis work before shifting to full-time staff in 1999. Phillips is the lead developer of the scalable molecular dynamics program NAMD, which received a 2002 Gordon Bell award and has been cited over 500 times. His research interests span the complete 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 the 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. Hao Yu 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 Kale Department of Computer Science, Thomas M. Siebel Center, University of Illinois at Urbana-Champaign, Urbana, IL 61801. (kale@uiuc.edu). Professor Laxmikant Kale has been working on various aspects of parallel computing, with a focus on enhancing performance and productivity via adaptive runtime systems, and with the belief that only interdisciplinary research involving multiple CSE and other applications can bring back well-honed abstractions into Computer Science that will have a long-term impact on the state-of-art. His collaborations include the widely used Gordon-Bell award winning (SC'2002) biomolecular simulation program NAMD, and other collaborations on computational cosmology, quantum chemistry, rocket simulation, space-time meshes, and other unstructured mesh applications. He takes pride in his group's success in distributing and supporting software embodying his research ideas, including Charm++, Adaptive MPI and the ParFUM framework.

L. V. Kale received the B.Tech degree in Electronics Engineering from Benares Hindu University, Varanasi, India in 1977, and a M.E. degree in Computer Science from Indian Institute of Science in Bangalore, India, in 1979. He received a Ph.D. in computer science in from 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.