Enabling and Scaling Biomolecular Simulations of 100 Million Atoms on Petascale Machines with a Multicore-optimized Message-driven Runtime

Chao Mei, Yanhua Sun, Gengbin Zheng,
Eric J. Bohm, Laxmikant V. Kale
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA
{chaomei2, sun51, gzheng, ebohm, kale}@illinois.edu

James C.Phillips, Chris Harrison
Beckman Institute
University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA
{jim, char}@ks.illinois.edu

ABSTRACT

A 100-million-atom biomolecular simulation with NAMD is one of the three benchmarks for the NSF-funded sustainable petascale machine. Simulating this large molecular system on a petascale machine presents great challenges, including handling I/O, large memory footprint and getting good strong-scaling results. In this paper, we present parallel I/O techniques to enable the simulation. A new SMP model is designed to efficiently utilize ubiquitous wide multicore clusters by extending the CHARM++ asynchronous message-driven runtime. We exploit node-aware techniques to optimize both the application and the underlying SMP runtime. Hierarchical load balancing is further exploited to scale NAMD to the full Jaguar PF Cray XT5 (224,076 cores) at Oak Ridge National Laboratory, both with and without PME full electrostatics, achieving 93% parallel efficiency (vs 6720 cores) at 9 ms per step for a simple cutoff calculation. Excellent scaling is also obtained on 65,536 cores of the Intrepid Blue Gene/P at Argonne National Laboratory.

1. INTRODUCTION

Biomolecular simulations are highly challenging to efficiently scale to a large number of processors. To be sure, they exhibit an abundant degree of parallelism. However, the challenge arises from the size of each time step and the number of time steps needed for interesting simulations. Due to the atomic-level time and length scales being modeled (e.g. the vibration frequency of bonds), each time step can only be of the order of 1 fsec. At the same time, the biological phenomena of interest to require simulations of microsecond or longer duration. Thus, a meaningful simulation needs to carry out billions of individual time steps. This means that each individual timestep must be carried out in milliseconds: at 10 ms per step, and even using a 2 fsec per time step, we would need two months to complete a microsecond simulation. It is highly challenging to get individual time steps confined to that small a time interval, considering the variety of computations that must happen during this time and the communication dependencies between them. Issues related to load imbalances, critical paths, communication-computation overlap, progress issues for the underlying communication engine (i.e. MPI runtime) etc. create obstacles to obtaining high-performance.

In this context, the new generations of machines with hundreds of thousands of processors pose new challenges. With these machines, the goal of simulating molecular systems with tens of millions of atoms while still maintaining execution time-per-step within tens of milliseconds appears to be within reach. In fact, the National Science Foundation posed such an ambitious goal in the requirements for the track one machine: namely, to simulate a 100 million atom system with a time-per-step of the order of 10 ms. This paper focuses on this challenge, and the solution techniques being developed to overcome it.

As machines become more powerful, the size of biomolecular system that can be studied through all-atom simulation has increased exponentially. In 2006, NSF introduced a 100-million-atom biomolecular simulation as one of the three benchmarks for the NSF-funded sustainable petascale machine. Such specific-application petascale benchmarks in the HPC acceptance process are a welcome addition for the community, because they apply realistic performance constraints on both harware and software beyond those in LinPACK and artificial microbenchmarks.

A relatively new issue that biomolecular simulation programs are now required to tackle is that presented by multicore nodes. Typical machines today have between 12 and 48 cores per node. Historically, efficiently exploiting shared memory within a node has proved difficult, and many applications continued using separate processes on each core. However, the benefits to be reaped in exploiting shared memory remain compelling; and in any case for a large molecular system such as the one targeted in this paper, it is necessary to exploit it in order to fit within the available memory.

Introducing support for unusually large systems into a long-lived application presented several challenges, including the traditional problem of achieving good strong-scaling results, as well as new ones in I/O due to the problem size. Through the application of a variety of techniques to both the application, NAMD, and the underlying runtime system, CHARM++, we reduced per-core and per-node memory footprint and resolved a number of performance bottlenecks to achieve strong scaling for a 100-million-atom system to the full Jaguar PF machine.

Contributions. We present both enabling and scaling techniques that are required to simulate a 100M-atom system on machines of

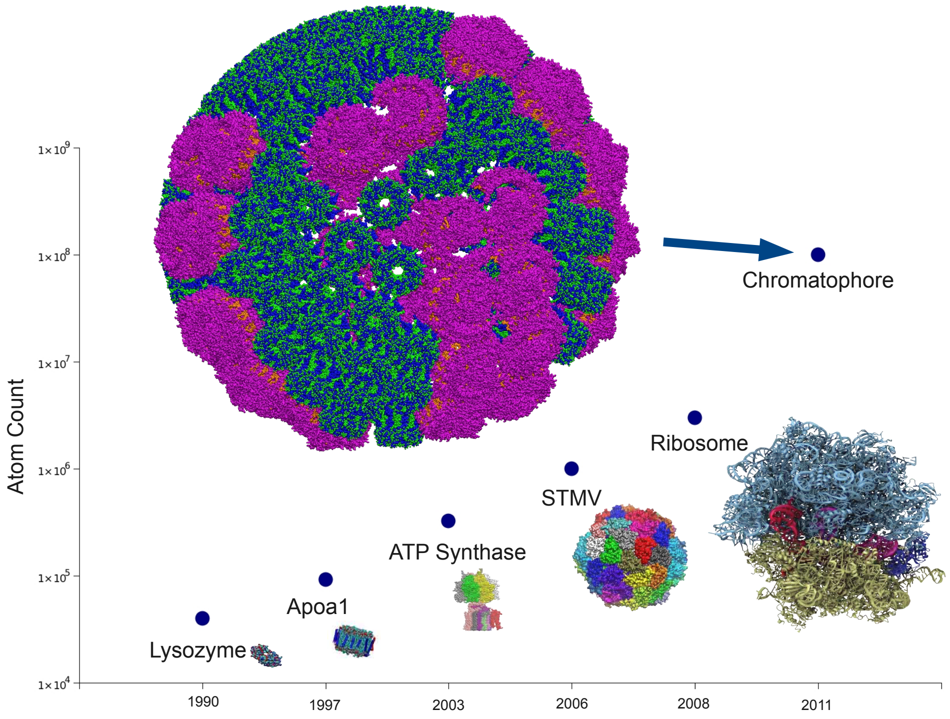


Figure 1: The size of biomolecular systems that can be studied using all-atom molecular dynamics simulations has steadily increased from that of Lysozyme (40,000 atoms) in the 1990s to the F_1F_0-ATP Synthase and STMV Virus capsid at the turn of the century, and now 100-million atoms as in the spherical chromatophore model shown above. Atom counts include aqueous solvent, not shown.

up to 224,076 cores. Our key contributions are:

- We present enabling techniques based on parallel I/O to dramatically reduce memory footprint and speed up NAMD initialization phases. After memory optimizations, the 100M-atom system can run on Intepid (Blue Gene/P), which has only 0.5GB of memory per core.
- The CHARM++ asynchronous message-driven runtime is extended with a new SMP mode, which is designed to efficiently utilize the ubiquitous wide multicore environments.
- We present a series of node-aware optimizations enabled by the SMP mode of runtime to improve the scalability of NAMD. These techniques include node-aware communication optimizations, and further optimized hierachical load balancing strategy that scales to 224,076 cores.
- We scale the 100-million-atom simulation up to the full Jaguar PF system (224,076 cores) at Oak Ridge National Laboratory with impressive benchmark of 9.00 ms/timestep for cutoff run w/o barrier, 16.84 ms/timestep with cutoff run w/ barrier, and 26.26 ms/timestep with PME run. A parallel efficiency of 93% based on 6720-core run is achieved for cutoff run w/o barrier on the full Jaguar PF system, along with an excellent speedup up to 65,536 cores on Intrepid Bleue-Gene/P machine at Argonne National Laboratory.

The remainder of the paper is organized as follows: Section 2 describes the background of NAMD and the CHARM++ runtime system. Section 3 explores the parallel I/O techniques to overcome the memory challenges during NAMD start-up. The design and node-aware optimizations of the SMP mode of CHARM++ runtime for multicore clusters are presented in Section 4. Performance results of strong scaling of a 100-million-atom simulation are provided in Section 5, as well as weak scaling results on both Jaguar and Intrepid. Other scalable MD codes are discussed in Section 6. Finally, Section 7 concludes the paper with some future plans.

2. BACKGROUND

In this section, we briefly describe the biomolecular simutation program NAMD and its parallelization techniques using the underlying runtime CHARM++.

2.1 Biomolecular Simulation

NAMD [17] is one of a class of programs that perform classical, all-atom simulations of biopolymers in explicit solvent. For such simulations, initial coordinates of proteins are determined through X-ray crystallography or other experimental techniques, combined with lipids or nucleic acids, and solvated in water molecules and ions to fill a periodic simulation cell. Interactions between atoms in the simulation are parameterized based on the species of each atom and its chemical role (i.e., its covalent bonding pattern). Covalent bonds are represented by harmonic distance, angle, and planarity restraints and sinusoidal dihedral terms. Pairs of atoms that are not bonded to each other or to a common atom interact via a nonbonded potential that is a combination of a Lennard-Jones 6-12 potential and electrostatic interactions between fixed partial charges assigned to each atom to reflect its relative electronegativity. The Lennard-Jones nonbonded term is smoothly truncated at a cutoff distance, typically 12 Å, while the electrostatic potential is divided into short-range interactions with the same cutoff distance and long-range interactions extended to all periodic images via the FFT-based particle-mesh Ewald method (PME). The negative gradient of the complete potential defines forces on all atoms that are integrated by the explicit, reversible, and symplectic Verlet algorithm to simulate the dynamic evolution of the system with a timestep of 1 fs. The communication-intensive PME method calculates only small and slowly varying forces, and may thereforce be evaluated only every forth step by using a multiple-timestep integrator. Modifications to the integrator allow simulation in NVE, NVT, or NPT ensembles, with pressure control requiring a periodic global barrier to modify the volume of the periodic cell.

As illustrated in Figure 1, biomolecular simulations have grown larger over time. Each increase in simulation size capability has permitted new discoveries unanticipated by smaller scale simulations. The extension of simulation size to hundreds of millions of atoms will permit all-atom simulations of extensive biomolecular complexes in key cellular systems, and even atomic models of entire organelles of a cell.

The 100-million-atom benchmark in this paper was assembled by replicating a million-atom satellite tobacco mosaic virus (STMV) simulation on a 5x5x4 grid. This provides a realistic input set for scaling NAMD to petascale systems without waiting for our biophysics collaborators to assemble a complex non-reptitive system of equivalent size, such as the chromatophore. This approach also simplifies correctness testing and the construction of smaller systems for weak-scaling studies. In this paper, three configurations are studied: simulation with PME full electrostatics, cutoff-only simulation with global barrier for each step, and cutoff-only simulation without global barrier. Both common and configuration-specific performance issues are investigated.

2.2 NAMD Parallelization with Charm++

CHARM++ [13] is a parallel programming system based on a message-driven migratable-objects programming model. In this model, the programmer decomposes his application into fine-grain objects that perform the computation and communicate through asynchronous method invocation by sending each other messages. Over the years, it has been successfully used to develop several highly scalable parallel applications, such as NAMD [2], ChaNGa [11], and OpenAtom [4]. Furthermore, CHARM++ is a highly portable parallel runtime system available on the vast majority of existing parallel platforms.

2.2.1 CHARM++ *on Multicore Clusters*

The CHARM++ runtime system provides several mechanisms for exploiting shared memory in the context of multicore machines [16]. Each platform specific implementation of CHARM++ can be built with an "smp" extension that allows the process space to be redefined from one flow of control per process to multiple flows per process, called "worker threads". The notional "rank" space is partitioned across all worker threads as it would have been across individual processes in the non-smp case. Worker threads, typically implemented via pthreads, share their parent process's address space, but contain their own event scheduler and appear semantically as independent ranks with a persistent mapping of chares to ranks (unless those chares are explicitely migrated). These threads are then typically affinitized to one thread per core and persist for the life of the application.

This model differs from hybrid programming models, such as MPI with OpenMP [20], in that it does not require a hybrid programming approach. The CHARM++ message driven object approach works for SMP and non-SMP versions without any SMP specific programmatic changes. Shared memory benefits such as reductions to overall memory footprint, reduced memory bandwidth consumption, faster application launch, and improved the efficiency of communication, can now be realized by CHARM++ applications, with no shared memory specific application code, on petascale machines as shown in Section 4.1.

In previous work [16], we have extensively studied multi-threading performance issues and techniques focusing on optimizing intra-node communication in CHARM++ applications running on single multicore desktops. This paper extends our optimization techniques including improving performance of inter-node communication for multicore clusters. We target such a multi-threaded runtime to petascale class machines and use real world application NAMD to demonstrate the optimization techniques we explored.

2.2.2 NAMD Parallel Decomposition

NAMD uses a hybrid parallel decomposition in which atomic coordinates and velocities are stored and propagated by static spatial domain objects called "patches" while the calculation of interactions between atoms is decomposed into independently migratable "compute objects". The patches are cubes (or similar shapes that fill the periodic cell) of dimension equal to the cutoff distance plus a sufficient margin that atoms in non-neighboring cubes will not move closer than the nonbonded cutoff distance during a migration cycle of typically 20 timesteps. Patches are represented on other processors by "proxies" through which compute objects access atomic coordinates and store forces. Atomic coordinates and forces are communicated only between a patch and its proxies. A compute object is generated for every pair of neighboring patches, as well as for bonded terms on each processor with patches.

2.2.3 Static and Dynamic Load Balancing

A challenging aspect of scaling biomolecular simulations to a very large number of processors is that computation work due to bonded and direct electrostatic force calculation, PME computation, and force integration, has to be distributed evenly across the whole machine. Furthermore, as atoms move, load imbalance may occur. To tackle this load balance problem, NAMD combines an initial mapping scheme at start-up time with dynamic load balancing to adjust load imbalance during the simulation.

The static mapping of patches to processors in the case of more patches than processors is based on ordering the patch grid by a continuous space-filling curve that varies fastest in the Z dimension and slowest in the X dimension, snaking up and down, back and forth, to fill the simulation volume from one end to another. The patches thus ordered are assigned to processors in contiguous groups so as to spread atoms evenly across the machine. The space-filling curve is designed to optimize communication between patches and the PME 3D FFT, statically decomposed either 1D via slabs (for small simulations or processor counts) or 2D via pencils (for large simulations on large processor counts). Block-wise mapping of processor indices to nodes ensures that only nearby patches are mapped both within processors and within nodes. At larger processor counts patches will be distributed across the machine with at most one patch per processor, and eventually patches divided in one, two, or three dimensions to increase parallelism.

Initial compute placement similarly optimizes communication, with each patch having at most seven proxies on the processors of its "downstream" neighbors. This pattern provides that every pair of neighboring patches is represented on at least one processor, to which the corresponding compute may be assigned. Patch and compute object assignments are replicated in per-process patch map and compute map objects, both using storage proportional to the size of the simulation for the sake of efficient access.

NAMD exploits a measurement-based load balancing method supported in CHARM++ for balancing computation across processors. The NAMD load balancer attempts to create the smallest number of additional proxies needed to achieve load balance. Traditionally, before the work in [22], NAMD uses strategies that collect all load statistics to a central location (typically processor 0), where load balancing decisions are made. This scheme is not scalable, as runtime and memory usage increase with both processor and patch count. In [22], we explored a hierarchical load balancing scheme with a 1-million-atom STMV system on up to 16,384 cores of Intrepid (Blue Gene/P). This paper extends the work to scale to the full Jaguar PF machine, currently the fastest US supercomputer and second in the world.

2.3 Targeted Petascale Machines

For this 100M-atom scaling study we selected two petascale machines. The first is Intrepid, a 557.1 TF/s, 40 rack IBM Blue Gene/P at Argonne National Laboratory. The second is Jaguar PF (later referred just as Jaguar), a 2.3 PF/s Cray XT5 at Oak Ridge National Laboratory. Each node on Intrepid has 4 cores and only 2GB of memory, with a total of 40,960 compute nodes/163840 cores. In comparison, Jaguar contains 18,688 compute nodes, or 224,256 cores with each multicore compute node containing dual hex-core AMD Opteron processors, and 16GB of memory. Both machines have a relatively low memory ratio – 0.5GB per core on Intrepid and 1.3GB per core on Jaguar. Of the two machines, Jaguar is closer in size to Blue Waters, the ultimate target of our efforts.

3. APPLICATION MEMORY CHALLENGES AND TECHNIQUES

Due to the number of atoms in this benchmark and its intended scaling to hundreds of thousands of processors, memory consumption becomes a critical issue. Initial experiments showed that on machines such as BlueGene/P that have small memory per core, NAMD failed to pass the initialization phases due to the large memory footprint required to load the system and perform initialization. In this section, we will focus on two major memory issues from the aspect of application I/O, i.e., loading molecular data at start-up and outputing atoms trajectory data to the file system. Some other memory issues, which we overcome by utilizing the underlying multi-threading runtime, will be covered later in Section 4. Existing parallel I/O libraries such as HDF does not handle NAMD file formats, therefore we chose to implement parallel I/O natively in NAMD. One advantage is that we can then optimize for writing trajectory data frame-by-frame, overlapping with a running simulation.

3.1 Parallelizing Molecular Data Input

Traditionally, NAMD performed initialization by first loading all molecular data and processing it on a single core before broadcasting the data to other cores. Although adequate when the molecule size is small, this approach does not scale due to the sequential bottleneck. For example, a test of the 100M-atom system on a 4-socket Intel Xeon L7555@1.87GHz workstation with 64GB physical memory, required 3301.91 seconds and 40.48GB of memory for initialization. Using the compression scheme we introduced in [2], the initialization time drops to 125.47 seconds, but it still required 12.82GB of memory. Since Jaguar has 16GB per node shared by 12 cores and Intrepid has only 2GB per node shared by 4 cores, reducing the memory footprint of initialization is critical.

Therefore, it is clear that the initialization process needs to be parallelized to distribute the memory usage and speed up the process. We designed a parallelization scheme for NAMD on top of the compression scheme described in [2], where the "signatures" of atoms are extracted from input data to represent the common characteristics that are shared by a set of atoms. In addition to the "signature" file, a binary file that contains the information of each atom is constructed from the original input file to be used for parallel input.

Given P input processors, a free parameter automatically tuned to optimize footprint and performance, one of them will first read the signature file and then broadcast this information to all input processors. For this 100M-atom simulation, the signature file is about 114KB, 5800 times smaller in size than the original input file that was required to be loaded from one processor. Afterwards, each of these P cores will load $\frac{1}{P}$ of total atoms starting from independent positions of the binary file. Finally, some atoms are shuffled with neighbor input processors according to molecular grouping attributes for later spatial decomposition.

The parallelization of input has dramatically decreased startup time and memory usage. In contrast to the original version of NAMD, which takes nearly an hour to finish initialization and more than 40GB of memory for this 100M-atom simulation, the optimized version with 600 input processors on Jaguar completes initialization in 12.37 seconds, with an average 0.19GB of memory consumption per input processor. Furthermore, compared to the old compression scheme, this is a factor of 10 and 67 decrease in initialization time and memory usage, respectively, thus making it possible to run the 100 million atom simulation in NAMD on currently available petascale machines.

3.2 Parallelizing Trajectory Data Output

Trajectory and restart output is necessary for production biomolecular simulations. Performance and memory footprint issues are analogous to those faced at input, but maintaining fast timestep execution under tens of milliseconds poses an additional challenge for frequent trajectory output. For the 100M-atom system, the 4.8GB data per output step must not delay the simulation itself. To overcome those issues, we parallelized the output process so that each output core, a quantity usually smaller than the total number of cores, is only responsible for the trajectory output of a subset of atoms, together with the design of overlapping file output on one core with useful computation on other cores. The latter is naturally enabled by the asynchronous message-driven CHARM++ runtime.

For backward compatibility with our visualizer component VMD [10] , we initially chose to output the data into a single file in the old file format. To avoid the contention on file I/O accesses by simultaneous writers, we implemented a token-based output scheme in which only the processor that has the token could write to the file system. However, with experiments on Intrepid and Jaguar, we found that the performance of writing into a single file is not acceptable. Some experiments were conducted with 2.8M-atom Ribosome simulation to illustrate the issue we identified. Figure 2 shows excerpts from tracing results of output processors for parallelizing output on 32 nodes of Jaguar with one output processor on each node, where the orange bar represents the operation of file I/O. First, (a) and (b) clearly show the unacceptable performance of writing into a single file. In (a), the output clearly overlaps the useful computation (in blue color), but spans across multiple timesteps. In (b), the simultaneous output, though into different non-overlapped portion of the file, showed a severe stretch on file I/O. In this case, the simultaneous output took even more time (2.94 sec vs. 1.79 sec) than that in the case shown in figure2(a). This left us a surprising and counter-intuitive impression of the underlying parallel file system in that users were supposed to obtain a linear speedup over the single writing token scheme. Based on document[1], we believe such behavior is caused by contention on

[1]http://www.nics.tennessee.edu/io-tips

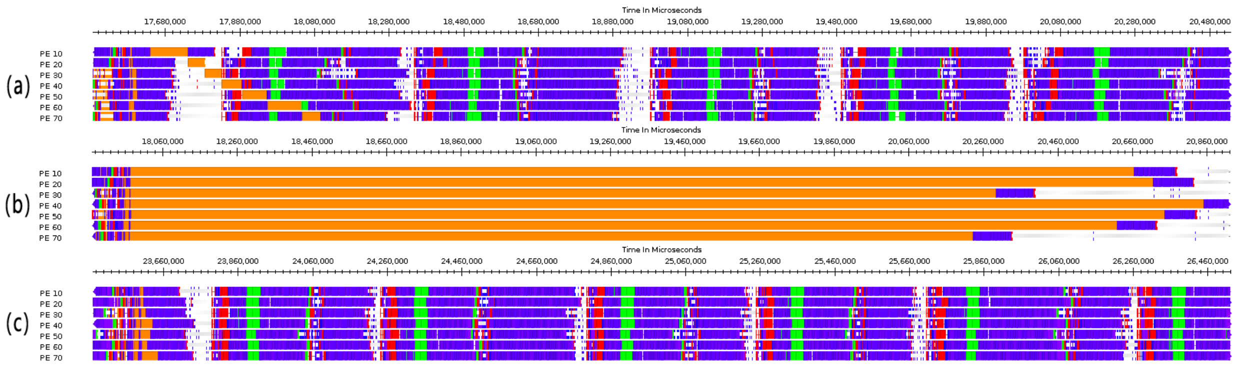


Figure 2: Output strategy timelines on Jaguar: (a) One output processor writes at a time (1 token), (b) All write simultaneously (32 tokens), (c) All write simultaneously but to multiple independent files.

updating the meta-data on that file system.

As a result, we resort to outputting data into multiple independent files, and then post-processing them into a single file in the original format. Figure 2(c), at the same scale with figure 2(a) and (b), clearly shows the huge benefit of this approach, reducing the output time by orders of magnitude to 0.048 sec. One will notice that the single output by one core in the multi-file-output scheme even takes less time than that of the 1-token based scheme. This is because the the output data layout is the same with layout of these data in memory in the multi-file-output scheme.

Future optimizations are possible for both input and output in NAMD to achieve even better performance. However, the current implementations are good enough to minimally impact the overall actual simulation performance and are fully portable across all platforms of interest.

4. NODE-AWARE OPTIMIZATIONS FOR MULTICORE CLUSTERS

When we started our effort to scale NAMD with the 100M-atom system to these two petascale machines, we encountered several issues using the production version of NAMD that runs one process per core (i.e. non-SMP mode).

Even with the parallel I/O techniques described in Section 3, NAMD ran out of memory when simulating the 100M-atom system on Intrepid. On Jaguar, the initial performance is shown in Table 1. Both PME and cut-off simulations do not scale well.

Nodes	140	560	2240	4480	8960	17920
Cores	1680	6720	26880	53760	107520	215040
PME	1295.5	351.2	111.2	60.3	39.6	45.5
Cutoff	1097.3	329.5	133.3	68.7	49.1	38.3

Table 1: Initial benchmark time with PME and w/o PME on Jaguar running with non-SMP runtime

Performance analysis showed that the major challenges preventing NAMD from scaling on these two machines were memory footprint, system noise, communication bottleneck, and load imbalance. Our contribution in this paper is that by exploiting optimization techniques for multicore architectures in both the runtime and the application, we optimized the memory footprint and scaled NAMD with the 100M-atom system to massively parallel machines such as Jaguar. Note that the techniques we apply here are general, and they can be applied to other petascale applications as well.

4.1 A Multi-threaded MPI-based Runtime

In order to exploit the multi-threaded SMP model in CHARM++ on Jaguar, we designed and implemented an SMP runtime mode on top of MPI, the de facto standard of communication interfaces for high-performance computers. In this particular case, the runtime is built on the default MPI library on Jaguar implemented by Cray and POSIX threads. In SMP mode, the processor cores within a host operating system image are mapped either to worker threads (CHARM++ "processors" running as a single thread within a CHARM++ "node" process) or to a single communication thread per process. Threads within the same node process communicate via their shared address space while inter-node communication is handled by the communication threads.

Although each thread can independently call MPI functions to send and receive messages, this scheme is not chosen due to the way MPI supports threads. The MPI standard defines four levels of increasing thread support as MPI_THREAD_SINGLE, ...FUNNELED, ...SERIALIZED, and ...MULTIPLE. Many MPI implementations, including the default one we used on Jaguar, however, do not support multiple threads calling MPI concurrently. Even though some MPI implementations support the highest level of multi-threading, we observed that they do not deliver good performance due to implementations that have to ensure thread-safe for lower-level communication libraries. Furthermore, assuming the lowest level of MPI thread support ensures that our SMP model runs on top of most MPI implementations.

Therefore, we chose to shift all MPI functions to the communication thread. When a worker thread sends a network message, it enqueues the message to the communication thread's outgoing message queue; when an incoming network message arrives, communication thread receives the message and puts it into the corresponding worker thread's incoming message queue.

In practice, each worker thread and communication thread should be mapped to one physical core of a node to avoid interference with each other. Therefore, one core on the node is dedicated to the communication thread. The loss of one core for computation may degrade performance such that SMP mode may underperform non-SMP mode, expressed as:

$$P * E_{nonsmp}(P) > (P-1) * E_{smp}(P-1)$$

where P is the total number of cores per node, $E_{nonsmp}(P)$ is the parallel efficiency[2] in non-SMP mode, and $E_{smp}(P-1)$ is the parallel efficiency in SMP mode. Even in the best scenario when

[2]Parallel efficiency is $E(P) = \frac{T_1}{P*T_p}$

the parallel efficiency in SMP mode is 1, as long as the parallel efficiency in non-SMP mode satisfies:

$$E_{nonsmp}(P) > \frac{P-1}{P}$$

the non-SMP mode has better performance than the SMP mode. Take Jaguar where there are 12 cores per node as an example, if the parallel efficiency of non-SMP mode is greater than $11/12$, then the non-SMP mode always performs better than the SMP mode. However, the work associated with communication must be borne in either case.

If the parallel efficiency in non-SMP mode drops faster, and the ratio against the parallel efficiency in SMP mode turns below the above threshold, the SMP mode starts to perform better. Such scenario could happen when the application's communication-to-computation ratio increases. In non-SMP mode, worker threads have to pay the overhead of communication themselves, while in SMP mode the communication threads offload the communication from worker threads. However, when the communication thread can not keep up with the network communication that is requested by all the worker threads it serves, it becomes a performance degrading bottleneck. One possible solution is to reduce the ratio of worker threads to communication threads. For example, in the case of Jaguar, instead of having one communication thread serving 11 worker threads, we can reduce the load of a communication thread by having one communication thread serving 5 worker threads in one process with two processes per node.

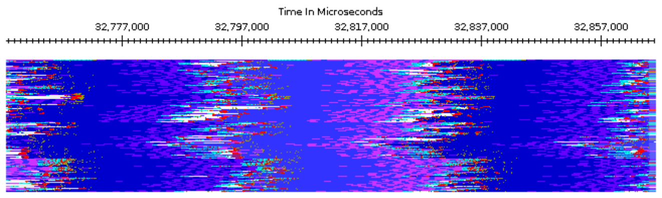
When designing this multi-threading SMP mode, we also considered processor affinity, i.e., the way threads are mapped to cores because it has been demonstrated to have a great impact on application's performance for multicore [16, 1, 18]. Taking account of the study [3] that shows system noise degrades NAMD's performance when using all 12 cores on each node on Jaguar, we bind the communication thread to core number 0, the most noisy core on the node where OS daemons are running, so that worker threads can execute without OS jitter. The worker threads are then pinned to remaining physical cores on the node respectively without differentiation. Such affinity setting assumes the application has a favorable computation to communication ratio, which is true for NAMD under most conditions.

Although this SMP mode is based on MPI, it can also apply to other low-level communication libraries such as LAPI, DCMF (for Blue Gene/P) and their descendants.

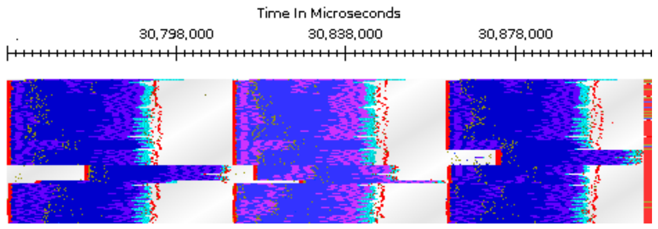
4.2 Adaptive Overlap of Communication and Computation

As analyzed before, in SMP mode, the communication thread may become overloaded to degrade the overall performance of NAMD. The over-decomposition and asynchronous communication in CHARM++ help to alleviate this potential problem. First of all, when one object is idle waiting for message arrival on a processor, another object on the same processor could exploit this idle time by doing useful work. Furthermore, asynchronous communication, such as asynchronous broadcast and reductions implemented on top of MPI, allows adaptive overlap of communication with computation. Therefore, the occasional delay in message delivery by the communication thread can be effectively hidden by the worker threads doing their useful work.

For example, Figure 3(a) shows a timeline view of the simulation of the 100M-atom system on 45,056 cores of Jaguar obtained by the performance analysis tool Projections [14]. In the figure, each line represents the activity of one processor over the time. Different work is shown in different colors. The red (dark) corresponds



(a) Timeline of cutoff simulation without pressure control barrier



(b) Timeline of cutoff simulation with pressure control barrier

Figure 3: Impact of barriers on 45056 cores of Jaguar

to the force integration, while the dominant blue-colored regions represent non-bonded computation. White-colored regions are idle time caused by the delay in the arrival of messages. Nevertheless, one can notice that neighboring timesteps, separated by dark red, ”bleed” into each other as a result of no global barrier after each timestep, nicely demonstrating the overlap of communication and computation. So even if the communication is overloaded in this case delaying the message deliver, the overall impact on performance remains small.

However, if a global barrier enforces, delayed messages caused by overloaded communication threads could no longer be hidden by the computation as clearly seen in figure 3(b). As a result, the performance becomes worse.

4.3 Benefits due to Multi-threading Runtime

With multi-threading in the runtime, in addition to the faster intra-node message (as no copy is involved either on sender or receiver side), the following extra benefits are observed and demonstrated by NAMD with this 100M-atom simulation on Jaguar.

Faster start-up: One nice side effect of using SMP mode is that it significantly reduces the job launching time because only one MPI rank (i.e., one instance of MPI communication library) is created for the entire node and multiple threads are spawned for each physical cores on the node. In contrast, in non-SMP mode, one MPI rank is created for each physical core. In the case of running on Jaguar, this means a 12X reduction in the number of total MPI processes launched as each node has 12 physical cores. Such reduction causes more significant launching time decrease when NAMD scales to 224,076 cores. In SMP mode, **mpirun** only takes about 1 minute to launch the job, while in non-SMP node, it takes about 6 minutes.

Further reduction in memory: That worker threads, i.e., “processors” in the common concept, on the same node shares the same virtual memory address space permits them to share read-only data structures. The memory footprint of the application will be reduced further by utilizing this runtime feature of CHARM++. Ignoring the feature will not break correct CHARM++ programs, but potential benefits related to memory effects such as better cache performance will be lost. In NAMD, optimizations are made to share the read-only information, such as the molecule object that contains static physical attributes of atoms, map objects that track the distribution of patch and compute objects. Table 2 shows the comparison of av-

#Nodes	140	560	2240	4480	8960	17920
#Cores	1680	6720	26880	53760	107520	215040
non-SMP (MB)	838.09	698.33	798.14	987.37	1331.84	1760.86
SMP (MB)	280.57	141.83	122.41	126.03	131.84	157.76
Reduced factor	2.99	4.92	6.52	7.83	10.10	11.16

Table 2: Comparison of average memory footprint between SMP and non-SMP during simulation (12 cores per node)

erage memory usage per core when running NAMD in non-SMP and SMP modes, demonstrating the effectiveness of reducing the memory consumption using SMP mode.

According to table 2, we first observe that the memory usage of each mode will first decrease then increase with the increase of nodes. This trend is caused by a mix of three factors: a) whole input data distributed on every core is reduced with the increase in the number of cores used; b) the memory usage of some data structures, such as the map objects, grows linearly with the increase of cores; c) runtime, including the MPI library, also requires more memory when scaling up.

Directly related with memory footprint reduction, we also observed much better cache performance. For example, using the PAPI [6] counters available on Jaguar, we witnessed a significant decrease in L1 data cache misses in SMP mode over non-SMP mode for each timestep as 63.91% and 78.18% on 4480-node and 8960-node run respectively. In addition, a slight 1.36% decrease in L2 cache misses is observed for the 4480-node run, and a decent 18.73% reduction for the 8960-node run.

4.4 Communication Optimization

Multicore clusters present hierarchical communication structure that includes both the intra-node communication, and inter-node communication, which is much more expensive. In both NAMD and its underlying runtime CHARM++ under SMP mode, taking advantage of the node level communication, we could effectively optimize communication by reducing the number of network messages (i.e. messages among different physical nodes).

4.4.1 Node-aware Communication

Node-aware multicast/broadcast in CHARM++: In the SMP mode of runtime, sending a multicast message to a subset of cores on the same node can be optimized by sending only one message to the communication thread, and letting the communication thread forward the message to the destination cores. Compared with the non-SMP case, this may reduce the network messages if the subset of destination cores is large.

As an example, a node-aware broadcast can greatly improve the NAMD start-up process. In a NAMD run on 17,920 nodes (215,040 cores) of Jaguar, the start-up phases involve a series of broadcast operations ranging from 4KB to 65KB. In non-SMP mode without node-awareness, it takes an average of 76.3 ms to finish, while in SMP mode it only takes an average of 20.2 ms, which is a speedup of 2.78.

Application-guided node-aware multicast spanning tree construction: NAMD multicasts atom data from each patch to all computes requiring that data at the begining of every timestep. NAMD takes advantage of the node-aware multicast operation (via spanning tree) supported by the runtime as mentioned above. However, given the fine-grained nature of NAMD (few ms per step), a second-order effect obstructs performance: a naive spanning tree construction may create an unbalanced spanning tree that overloads the intermediate spanning tree nodes which perform extra work of forwarding multicast messages down the tree. In NAMD we solve this issue by incorporating application level knowledge to build a balanced node-aware spanning tree that places heavily loaded processors in the leaves of the tree. The metrics we use to measure the load of each node is the number of patches and proxies on that physical node as it reflects the load of the communication thread doing the multicast. Based on this application-level knowledge, we sort the nodes that participate the multicast based on the total messages each transmits, and then construct the spanning tree based on this sorted node list.

4.4.2 Controlling Burst of Messages

Bursts of messages may occur at the beginning of each timestep of NAMD. On Jaguar, the flooding of network messages may lead to prolonged MPI_Iprobe calls as long as $12ms$ which causes shift of barrier as shown in Figure 3(b).

We modified the network progress engine in the communication thread to handle the burst of messages. In the network progress engine, the communication thread alternates among three types of tasks: (1) sending all the outgoing messages in the queue; (2) calling MPI_Test() on all the messages that have been sent and releasing the messages which are done; and (3) probing and receiving incoming messages until there are no more messages to receive. In the presence of message bursts, the communication thread can easily be delayed in one of these three tasks, and cause performance problems. To prevent the communication thread from stalling, we applied a cap on each of these tasks to control how many of each operation the communication thread can perform at a time. During the process of sending messages, if the communication thread has detected that it has sent out too many messages, or there are too many sent messages that have not been released by MPI library, then it will stop sending more messages and change to do other two tasks to avoid flooding the network. Similarly, when receiving messages from network, the communication thread will stop receiving more if it detects that there are too many outgoing messages in the queue.

We tested the new scheme in the case of the 100M-atom simulation. On large runs, where bursts become a problem, for example, on 4480 nodes (53,760 cores), we observed a performance improvement by 12.3% after applying this scheme.

4.5 Hierachical Load Balancing

In previous work [22], we have explored the hierarchical load balancing scheme with a 1 million STMV atom system on up to 16,384 cores of Intrepid. In this paper, we extend that work to support 100-million-atom systems and further optimized the hierarchical load balancing scheme for the full Jaguar machine (224,076 cores).

In the NAMD hierarchical load balancing scheme, every 512 cores form a load balancing group. Inside each group, centralized load balancing algorithms optimized for multicore are applied. One observation is that for the 100M atom system, the average load of each group is similar as shown in the bottom curve of Figure 4, therefore, we can disable the cross group load balancing to simplify and accelerate the load balancing process.

The effectiveness of the hierarchical load balancing scheme can be seen in Figure 4 for a 100M atom simulation of 53,760 cores of Jaguar. The X-axis represents total of 105 load balancing groups. The top curve is the maximum load of all the 512 cores in each corresponding load balancing group, and the bottom curve is the average load in each group, representing the best scenario if load balancing algorithm can achieve the load balance. The middle curve which almost overlaps with the bottom curve is the estimated maximum load of each group after the algorithm makes load balancing

Machine	Nodes	Cores	Cutoff w/o barrier			PME			Cutoff w barrier		
			Before ldb	After ldb	Improve(%)	Before ldb	After ldb	Improve(%)	Before ldb	After ldb	Improve(%)
Jaguar	560	6720	312.50	281.59	9.9	357.01	345.84	3.1	312.50	294.19	5.9
	8960	107520	24.20	18.62	23.1	39.19	36.49	6.9	31.85	28.74	9.8
	18673	224076	10.74	9.00	16.2	27.9	26.28	5.8	18.53	16.84	9.1
Intrepid	1024	4096	2305.7	2054.5	10.9	2653.0	2354.6	11.2	2331.4	2077.8	10.9
	4096	16384	636.6	530.7	16.6	729.3	606.1	16.9	643.6	536.6	16.7
	16384	65536	211.5	138.7	34.4	244.5	162.6	33.5	213.5	140.1	34.4

Table 3: Benchmark time (ms/step) before and after load balancing for runs on Jaguar and Intrepid

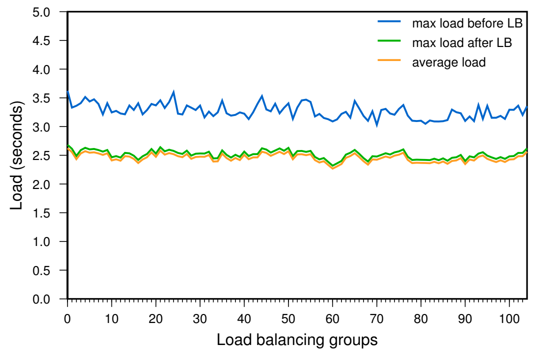


Figure 4: Load balancing effect on 53,760 cores of Jaguar

(a) Jaguar non-SMP and SMP

Cores	PME		Cutoff w/ barrier		Cutoff w/o barrier	
	non-SMP	SMP	non-SMP	Smp	non-SMP	SMP
1680	1295.5	1344.0	1097.3	1118.5	*	*
6720	351.15	345.84	329.51	294.19	319.53	281.59
53760	60.34	54.25	68.67	44.21	43.49	36.84
107520	39.58	36.49	49.10	28.74	25.07	18.62
224076	45.52	26.28	38.25	16.84	14.58	9.00

(b) Intrepid SMP

Nodes	Cores	Workers	PME	cutoff-barrier	cutoff-nobarrier
512	2048	1536	4754.7	4158.4	4031.6
1024	4096	3072	2354.6	2077.8	2018.5
4096	16384	12288	606.1	536.6	517.4
16384	65536	49152	162.6	140.1	138.7

Table 4: Performance (ms/step) of 100-million-atom simulation

decisions. This shows that the load balancing algorithm is very effective in reducing the maximum load to be close to the average.

Compared with the case of NAMD in non-SMP mode, using SMP mode improves the load balancing quality. This is because after communication threads offload all communication load from worker threads, the load balancer can obtain more accurate information about the measurement of actual computation load on worker threads. This potentially leads to better load balancing decisions since the load balancer does not have to consider the complicated scenarios when communication overhead is involved on worker threads.

5. EXPERIMENT RESULTS

In this section, we present load balancing results, overall strong scaling performance results of the 100M-atom simulation with three different configurations (PME, cutoff with barrier,cutoff without barrier) on Intrepid and Jaguar. We also provide the weak scaling performance results of simulations of smaller atom systems from multi-million to tens-of-million atoms.

5.1 Load Balancing Results

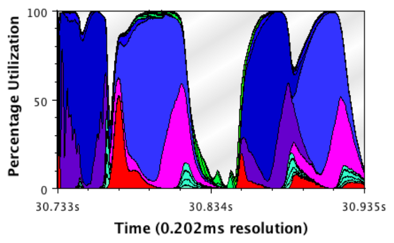
Table 3 shows the benchmark time for three types of 100-million-atom simulation before and after load balancing on Jaguar and Intrepid. One observation that is common for the three types of simulation is that on smaller number of cores (560 nodes on Jaguar and 1024 nodes on Intrepid), load balancing helps performance by less than 10%. This is because the static initial mapping already does a good job balancing the load, as the load is well balanced (the ratio of maximum load to average load is only slightly more than 1). On larger number of cores where computation becomes more fine grain and initial mapping is not as effective, load balancing improves performance by as much as 30% for cutoff without barrier simulation on Jaguar and for all the three types of runs on Intrepid. Among the three types of simulation, load balancing helps least in simulations with PME on Jaguar. This is because NAMD suffers from system noise on Jaguar, and the barrier involved in PME calculation makes it challenging to balance the load in the presence of the OS jitter.

Comparing these two different supercomputers, we find that hierarchical load balancing strategy performs better on Intrepid than that on Jaguar. This is probably because micro-OS kernel of the Blue Gene/P is less noisy [15]. Therefore, CHARM++ load balancer tends to make more accurate load balancing decisions.

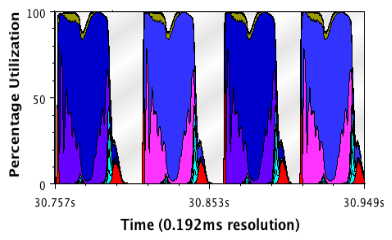
5.2 Strong Scalability Results

With all the techniques and optimizations in our SMP model, we have achieved excellent performance on Jaguar. Table 4 compares the benchmark time for three types of simulation of with PME, cutoff with barrier and cutoff without barrier in both non-SMP and SMP models on Jaguar and SMP on Intrepid. Non-SMP performs better than SMP on smaller number of cores (1680 cores) when the application utilization is relatively high. One core out of 12 dedicated as communication thread without doing real work does hurt performance. However, after the number of cores exceeds a threshold (6720 cores in our experiments), SMP performs better than non-SMP in all three types of simulation. The performance improvement of SMP over non-SMP on Jaguar rises in proportion to the number of cores, which is clearly seen from the overall speedup in Figure 5. Especially from 107,520 cores to 215,040 cores, speedup in non-SMP run for PME drops while the scaling continues in SMP. For the full Jaguar run of 224,076 cores, the speedup in SMP is almost twice that of non-SMP, which demonstrates the effectiveness of the SMP model and the optimizations.

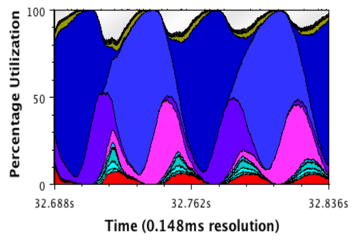
On Intrepid, the 100-million-atom simulation does not run without SMP due to its memory limit of 2G bytes per node. Based on 512 node performance, nearly perfect strong scalability results are achieved with SMP mode shown in Figure 6. Detailed numbers are presented in Table 4 (b). One issue is that dedicating one core to communication thread in SMP mode loses 25% of computing power. However, as discussed in Section 4.1, when the number of cores increases and the parallel efficiency decreases, SMP mode may perform closer or better than the non-SMP. Note that



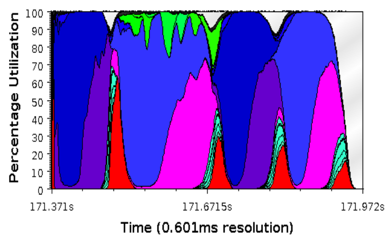
(a) Jaguar PME



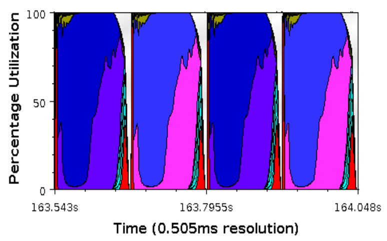
(b) Jaguar cutoff w/ barrier



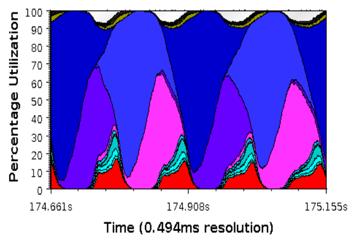
(c) Jaguar cutoff w/o barrier



(d) Intrepid PME



(e) Intrepid cutoff w/ barrier



(f) Intrepid cutoff w/o barrier

Figure 7: Processor utilization on 4096 nodes (45056 worker cores) of Jaguar and 16384 nodes (49153 worker cores) of Intrepid

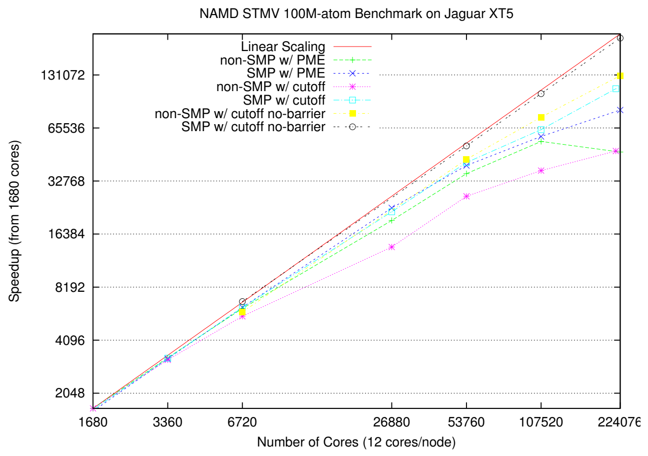


Figure 5: Speedup based on 1680 processors on Jaguar

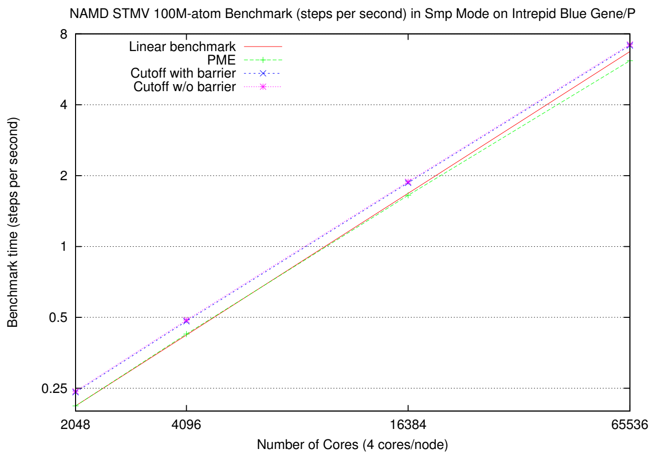


Figure 6: Performance (steps per second) on Intrepid

all near future supercomputers have a much larger number of cores per node, such as 32 cores on Power7-based supercomputers and 16 cores on Blue Gene/Q, each with an even higher number of hardware threads. Therefore, dedicating one core to communication thread is a viable choice.

To further understand the performance of the three types of simulation on these two machines, we examine the CPU utilization of all cores. Figure 7 shows the time profile for simulation with PME, with cutoff and with cutoff no barrier on 4096 nodes (45056 cores) of Jaguar and on 16384 nodes (49153 cores) of Intrepid. Common observations for the two supercomputers are that simulation with cutoff without barrier has the highest CPU utilization of about 93%, because it benefits from the adaptive overlap of communication and computation due to lack of barrier, while the CPU utilization is decreased by the 3DFFT transpose operations in PME simulations, and further degraded by the global barrier for pressure control. Comparing performance on the two supercomputers, higher CPU utilization is obtained on Intrepid than Jaguar for similar number of cores. Other than the fact that Intrepid has slower CPUs , the faster communication and lower CHARM++ runtime overhead [15] are important factors. Especially for the runs with cutoff and global barrier after each step, the performance on Jaguar suffers significantly. A detailed profiling for the CHARM++ MPI-based runtime on Jaguar revealed that MPI_Iprobe takes 12ms on some nodes, which significantly stretched the execution on these nodes. This suggests future work to improve the scalability of PME calculation, reduce the impact of system noise in the presence of global barrier, and explore platform specific optimizations to work around communication issues indicated by MPI_Iprobe.

5.3 Weak Scalability Results

Weak scalability, or how the computation time varies as a function of processor count for a fixed problem size per processor, was measured to study performance of NAMD for a wide range of simulation sizes that grow proportionally to the number of processors.

Systems of 2, 6, 12, 24, 48 and 100 million atoms were simulated using 341, 1024, 2048, 4096, 8192 and 16384 nodes, respectively, possessing four cores per node. Results, shown in Figure 8,

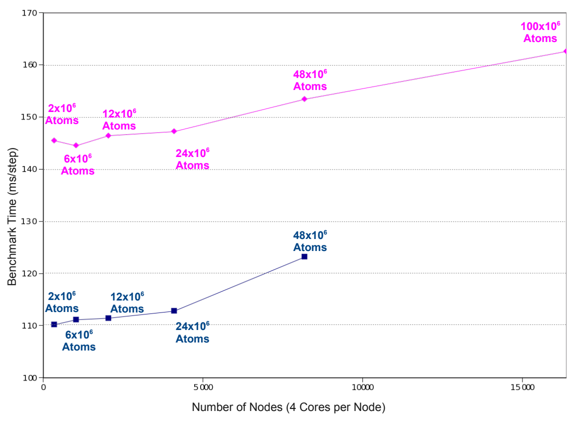


Figure 8: Weak scaling runtimes on Intrepid for SMP (top) and non-SMP (bottom) modes. 100M-atom system exceeds available memory per core in non-SMP case.

indicate that scaling up to 24M-atom systems on 4096 nodes maintains overall performance with only marginal increases of 2 milliseconds(2%) in compute time per MD timestep. Scaling of 48-million atoms to 8192 nodes leads to an approximate 7 millisecond jump (5%) in SMP, which is better than a jump of 12 millisecond (10%)in non-SMP cases. These benchmark time jumps per doubling of processors is likely due to communication, such as that involved in PME.

The 100-million-atom simulation only runs on 16384 nodes in SMP mode, thanks to all the optimizations employed to reduce memory footprint described in the paper. Note its performance is only about 3% increase of benchmark time comparing with the 48-million system on 8192 nodes, given that the 100M-atom system is about 4% larger in size than the ideal weak-scaling system size as 96 million atoms on 16384 nodes.

Per-processor performance is well maintained under weak scaling. The performance gap seen in Figure 8 between SMP and non-SMP corresponds to the loss of 25% computation power as we dedicate one core out of 4 per node to communication in SMP.

6. RELATED WORK

Other scalable MD codes such as Blue Matter [8], Desmond [5], Amber [7] and GROMACS [9] have not shown strong scaling results for such a large molecule system as NAMD has done in this paper. Work by Schulz R etc. [19] has shown how to scale multimillion-atom systems on Jaguar and presented a 100M peptide-water test system scaling up to 150K cores in a poster[3]. While we share the similar viewpoint that PME is the potential bottleneck on massively parallel supercomputers, our work actually demonstrates an actual 100M-atom biomolecule simulation with NAMD that scales upto 220K cores of full Jaguar machine with and without PME.

Some MD codes such as Desmond uses pthreads to implement a capability for each process to distribute its work among multiple threads in order to exploit multicore platforms. This MPI+pthreads type of hybrid programming requires the application developers to explicitly deal with multi-threading concurrency issues of threads. In general, compared with the popular hybrid programming approach including MPI+OpenMP [20, 12, 21] on SMP clusters, our SMP model is supported under the message-driven CHARM++ programming model. Application developers do not have to write shared memory specific code, whereas the runtime automatically optimizes message passing under the CHARM++ abstraction. Therefore, our approach is transparent to the application developers.

[3]http://cmb.ornl.gov/research/petascale-md/sc10_2.pdf

7. CONCLUSION

Biophysical molecular dynamics simulations present a vibrant research area capable of leveraging the opportunities presented by ever more powerful parallel machines. Providing application scientists with appropriate tools to explore larger biomolecular structures is a challenging task that we think is best met by an interdisplinary team willing and capable of examining the computational challenges at multiple levels, including application specific algorithms, communication runtime software, and hardware analysis.

In this paper, we demonstrated that a mature application can be extended to support a two order of magnitude increase in problem size. In the process of doing so, we showed that the application performance can be improved by leveraging awareness of physical locality within a node to reduce the memory footprint and to minimize intra-node communication. We also demonstrated the tradeoffs associated with different choices for the application of parallel input and output techniques in the context of a mature application.

Using the SMP mode of CHARM++, we demonstrated that the performance promise of shared memory multicore nodes can be achieved by careful threading at the runtime level. In doing so we demonstrated that this can be effectively utilized by a production application to improve strong scaling without any programmatic modifications. Combining the improvements in the runtime system with those in the application presents a powerful synergistic methodology for tuning applications to efficiently make use of the entirety of two different modern petascale machines with 93% parallel efficiency for strong scaling of a 100 million atom benchmark.

We have demonstrated application improvements which open up the possibility to apply all atom molecular dynamic techniques to study the evolution of structures with hundreds of millions of atoms. We anticipate that having applied these techniques to an application familiar to many thousands of end user scientists will facilitate deep and lasting insights. Though the resources to execute those investigations are currently limited to a handful of machines, the historical trends of high performance computing indicate that the power that is rare today is fated to become relatively common within a decade.

Acknowledgments

This work was supported in part by a NIH Grant PHS 5 P41 RR05969-04 for Molecular Dynamics, by NSF grant OCI-0725070 for Blue Waters deployment, by the Institute for Advanced Computing Applications and Technologies (IACAT) at the University of Illinois at Urbana-Champaign. This work used machine resources from NCCS (Jaguar), under allocation CSC076, supported by DOE, and Blue Gene/P (Intrepid) from Argonne National Laboratory, supported by DOE under contract DE-AC02-06CH11357. We also thank Osman Sarood who did the intial exploration of file I/O problems in enabling the 100M-atom simulation.

8. REFERENCES

[1] S. R. Alam, R. F. Barrett, J. A. Kuehn, P. C. Roth, and J. S. Vetter. Characterization of scientific workloads on systems with multi-core processors. In *In IISWC*, pages 225–236, 2006.

[2] A. Bhatele, S. Kumar, C. Mei, J. C. Phillips, G. Zheng, and L. V. Kale. Overcoming scaling challenges in biomolecular simulations across multiple platforms. In *Proceedings of IEEE International Parallel and Distributed Processing Symposium 2008*, pages 1–12, April 2008.

[3] A. Bhatele, L. Wesolowski, E. Bohm, E. Solomonik, and L. V. Kale. Understanding application performance via micro-benchmarks on three large supercomputers: Intrepid, Ranger and Jaguar. *International Journal of High Performance Computing Applications (IHHPCA)*, 24(4):411–427, 2010.

[4] E. Bohm, A. Bhatele, L. V. Kale, M. E. Tuckerman, S. Kumar, J. A. Gunnels, and G. J. Martyna. Fine Grained Parallelization of the Car-Parrinello ab initio MD Method on Blue Gene/L. *IBM Journal of Research and Development: Applications of Massively Parallel Systems*, 52(1/2):159–174, 2008.

[5] K. J. Bowers, E. Chow, H. Xu, R. O. Dror, M. P. Eastwood, B. A. Gregersen, J. L. Klepeis, I. Kolossvary, M. A. Moraes, F. D. Sacerdoti, J. K. Salmon, Y. Shan, and D. E. Shaw. 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.

[6] S. Browne, J. Dongarra, N. Garner, G. Ho, and P. Mucci. A Portable Programming Interface for Performance Evaluation on Modern Processors. *Int. J. High Perform. Comput. Appl.*, 14(3):189–204, 2000.

[7] D. A. Case, T. E. Cheatham, T. Darden, H. Gohlke, R. Luo, K. M. Merz, A. Onufriev, C. Simmerling, B. Wang, and R. J. Woods. The Amber biomolecular simulation programs. *J. Comput. Chem.*, 26(16):1668–1688, Dec. 2005.

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

[9] B. Hess, C. Kutzner, D. van der Spoel, and E. Lindahl. Gromacs 4: Algorithms for highly efficient, load-balanced, and scalable molecular simulation. *Journal of Chemical Theory and Computation*, 4(3):435–447, 2008.

[10] W. F. Humphrey, A. Dalke, and K. Schulten. VMD – Visual molecular dynamics. *Journal of Molecular Graphics*, 14(1):33–38, 1996.

[11] P. Jetley, F. Gioachin, C. Mendes, L. V. Kale, and T. R. Quinn. Massively parallel cosmological simulations with ChaNGa. In *Proceedings of IEEE International Parallel and Distributed Processing Symposium 2008*, pages 1–12, 2008.

[12] M. Jones, R. Yao, and C. Bhole. Hybrid mpi-openmp programming for parallel osem pet reconstruction. *Nuclear Science, IEEE Transactions on*, 53(5):2752–2758, oct. 2006.

[13] L. V. Kale and G. Zheng. Charm++ and AMPI: Adaptive Runtime Strategies via Migratable Objects. In M. Parashar, editor, *Advanced Computational Infrastructures for Parallel and Distributed Applications*, pages 265–282. Wiley-Interscience, 2009.

[14] L. V. Kale, G. Zheng, C. W. Lee, and S. Kumar. Scaling applications to massively parallel machines using projections performance analysis tool. In *Future Generation Computer Systems Special Issue on: Large-Scale System Performance Modeling and Analysis*, volume 22, pages 347–358, February 2006.

[15] S. Kumar, G. Dozsa, G. Almasi, P. Heidelberger, D. Chen, M. E. Giampapa, B. Michael, A. Faraj, J. Parker, J. Ratterman, B. Smith, and C. J. Archer. The deep computing messaging framework: generalized scalable message passing on the blue gene/p supercomputer. In *Proceedings of the 22nd annual international conference on Supercomputing*, ICS '08, pages 94–103, New York, NY, USA, 2008. ACM.

[16] C. Mei, G. Zheng, F. Gioachin, and L. V. Kalé. Optimizing a Parallel Runtime System for Multicore Clusters: A Case Study. In *TeraGrid'10*, number 10-13, Pittsburgh, PA, USA, August 2010.

[17] J. C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. D. Skeel, L. Kalé, and K. Schulten. Scalable molecular dynamics with NAMD. *Journal of Computational Chemistry*, 26(16):1781–1802, 2005.

[18] H. Pourreza and P. Graham. On the programming impact ofmulti-core,multi-processor nodes inmpi clusters. In *Proceedings of the 21st International Symposium on High Performance Computing Systems and Applications*, pages 1–, Washington, DC, USA, 2007. IEEE Computer Society.

[19] R. Schulz, B. Lindner, L. Petridis, and J. C. Smith. Scaling of Multimillion-Atom Biological Molecular Dynamics Simulation on a Petascale Supercomputer. *Journal of Chemical Theory and Computation*, 5(10):2798–2808, Oct. 2009.

[20] L. Smith and M. Bull. Development of mixed mode MPI / OpenMP applications. *Scientific Programming*, 9(2-3/2001):83–98, 2001.

[21] G. Tang, E. F. D'Azevedo, F. Zhang, J. C. Parker, D. B. Watson, and P. M. Jardine. Application of a hybrid mpi/openmp approach for parallel groundwater model calibration using multi-core computers. *Comput. Geosci.*, 36:1451–1460, November 2010.

[22] G. Zheng, A. Bhatele, E. Meneses, and L. V. Kale. Periodic Hierarchical Load Balancing for Large Supercomputers. *International Journal of High Performance Computing Applications (IHHPCA)*, 2010.