NAMD: A PARALLEL, OBJECT-ORIENTED MOLECULAR DYNAMICS PROGRAM

Mark T. Nelson
William . Humphrey
Attila Gursoy
Andrew Dalke
Laxmikant V. Kalé
Robert D. Skeel
Klaus Schulten

THEORETICAL BIOPHYSICS GROUP
UNIVERSITY OF ILLINOIS AND BECKMAN INSTITUTE
URBANA, IL 61801

Summary

NAMD is a molecular dynamics program designed for high performance simulations of large biomolecular systems on parallel computers. An object-oriented design implemented using C++ facilitates the incorporation of new algorithms into the program. NAMD uses spatial decomposition coupled with a multithreaded, message-driven design, which is shown to scale efficiently to multiple processors. Also, NAMD incorporates the distributed parallel multipole tree algorithm for full electrostatic force evaluation in $O(N)$ time. NAMD can be connected via a communication system to a molecular graphics program in order to provide an interactive modeling tool for viewing and modifying a running simulation. The application of NAMD to a protein-water system of 32,867 atoms illustrates the performance of NAMD.

Address reprint requests to Robert D. Skeel, University of Illinois at Urbana-Champaign, Theoretical Biophysics, 3111 Beckman Institute, 405 N. Mathews Avenue, Urbana, IL 61801; E-mail: skeel@cs.uiuc.edu

---

The International Journal of Supercomputer Applications and High Performance Computing,
Volume 10, No. 4, Winter 1996, pp. 251-268

1 Introduction

Molecular dynamics (MD) simulations (Brooks, Karplus, and Pettitt, 1988; McCammon and Harvey, 1987) play an important role in modern molecular biology. Widely used MD packages include CHARMM (Brooks et al., 1983), X-PLOR (Brünger, 1992), GROMOS (van Gunsteren and Berendsen, 1987), AMBER (Weiner and Kollman, 1981), and CEDAR (Carson and Hermans, 1985). MD simulations are very computer time intensive. Even simulations of systems of modest size, e.g., 5,000 atoms, require hours or days to complete. This limits the physical time and the size of the systems that can be studied. A promising means of overcoming these limitations is through the use of parallel computers. Several existing MD programs such as X-PLOR, CHARMM (Brooks and Hodošček, 1992), and GROMOS (Clark et al., 1994) have been altered to allow them to run on parallel machines. These programs contain a large body of sequential code that was modified to operate in parallel rather than redesigned specifically from the ground up for parallel execution. The program NAMD was, in contrast, developed specifically for distributed memory parallel machines. The program uses a spatial decomposition scheme to partition the domain in a way that provides maximum scalability. Independent threads of control are used to provide flexible load-balancing capabilities while maintaining a simple, uniform decomposition scheme. Message-driven scheduling is used to order the execution of these threads of control in a way that reduces the impact of communication latency. These principles lead to a high performance parallel design that scales well to large numbers of processors.

MD is a relatively new methodology, and many new techniques and algorithms are being developed that promise dramatic increases in the speed, size, and length of simulations that can be performed. In order to explore new methods with ease, NAMD uses an object-oriented design implemented in C++, an object-oriented extension to the C programming language. Its highly modular design and implementation allow new algorithms to be added easily without affecting the overall structure of the program. In addition, the design and implementation are documented so that NAMD can be understood by researchers without examining the source code.

A new algorithm that has been incorporated into NAMD is the distributed parallel multipole tree algorithm (DPMTA) (Rankin and Board, 1995). DPMTA provides an $O(N)$ means to calculate full electrostatics interactions, where N is the number of atoms in the simulation. Building upon previous work (Windemuth, 1995), DPMTA is incorporated into NAMD in an efficient and modular way via a multiple time stepping scheme and a set of interface objects.

NAMD can be employed within the interactive modeling system MDScope (Nelson et al., 1995), which allows researchers to view and alter a running simulation. MDScope links NAMD to a molecular graphics program via a set of communication routines and processes. This type of system is particularly useful for solving highly interactive modeling problems such as structure determination and refinement.

To demonstrate the performance of NAMD, it and two other MD programs are applied to the protein calmodulin in a sphere of water. To illustrate a demanding application of NAMD, the simulation of a protein-DNA complex—the estrogen receptor—in a bath of water comprising altogether more than 36,000 atoms is described. This system is large and needs full electrostatic interactions.

2 Features

NAMD is an MD package with the features necessary to perform typical molecular dynamics simulations. These features include the following: CHARMM19 and CHARMM22 parameter support, NVE ensemble dynamics, velocity rescaling, Langevin dynamics, harmonic atom restraints, energy minimization, and file compatibility with X-PLOR. Two forms of boundary conditions are currently provided by NAMD: one is vacuum, i.e., an infinite vacuum accessible to the model, and the other is a spherical boundary realized through harmonic potentials that restrain atoms within a sphere of a user-defined radius (Brooks and Karplus, 1983). Production quality simulations of several molecular systems are currently being performed with NAMD using these capabilities. Additional features such as NpT ensemble simulations, the ability to fix atom positions, and periodic boundary conditions are being developed. Also, NAMD has a couple of unique features that are detailed in the following subsections.

2.1 FULL ELECTROSTATICS USING DPMTA

The most computationally expensive operation in molecular dynamics is the computation of the nonbonded interactions. If a direct calculation method is used, the computation of electrostatic interactions between all pairs of atoms requires $O(N^2)$ operations. In order to avoid this large computational cost, most MD programs use cutoff distances, where interactions between pairs of atoms separated by more than the cutoff distance are neglected. However, it has been demonstrated that truncating the electrostatic interactions in this manner can qualitatively misrepresent physical properties of the system (Zhou and Schulten, 1995). In order to provide full electrostatic computations without incurring high computational overhead, NAMD employs DPMTA (Rankin and Board, 1995). This program implements a hybrid of the fast multipole algorithm (Greengard and Rohklin, 1987) and Barnes and Hut treecode algorithms (Barnes and Hut, 1986), and it reduces the computational complexity of evaluating electrostatic interactions for all pairs of atoms from $O(N^2)$ to $O(N)$. Like NAMD, DPMTA was developed for distributed memory parallel computers and scales efficiently to large numbers of processors.

2.2 INTERACTIVE MODELING

MDScope is a system developed to perform interactive MD simulations. It combines the computational power of NAMD with a molecular graphics program VMD using the communication package MDComm (Nelson et al., 1995). Such a system is invaluable for tasks such as structure refinement or structure determination. A sample setup for the use of MDScope is shown in Figure 1.

The high computational performance of NAMD is an essential requirement for MDScope, since effective interactive modeling requires that ten femtoseconds of simulation time be performed during each second of viewing time. To achieve such high computational rates, large parallel supercomputers are necessary. Thus MDScope enables VMD to connect from a remote graphics workstation to NAMD running on a parallel supercomputer.

The molecular graphics program VMD allows not only flexible viewing of static structures but also viewing and modification of running simulations. The key features of VMD are

- flexible selection language for choosing atom subsets for a variety of rendering and coloring options
- options for displaying images in stereo using a side-by-side format, or Crystal-Eyes stereo mode for suitably equipped systems
- support for use of spatial tracking devices that function as a three-dimensional pointer, with accompanying three-dimensional user interface in a stereo display environment
- modular design and implementation in C++.

MDComm is a set of library calls and processes developed at NCSA that allows communication between a graphical display program and a running MD simulation.

MDComm not only communicates results from the MD simulation but allows the graphics program to start a simulation and detach from and reattach to a running simulation. Key features of MDComm are

- process control in a networked environment
- support for heterogeneous systems
- concurrency on multiprocessor systems
- low-overhead implementation.

Currently, MDScope incorporates only limited capabilities for the modification of a running simulation. Development is under way to enhance these capabilities, e.g., to include the ability to switch between energy minimization and free dynamics; to place atoms, molecules, and side chains; and to apply forces to an atom or a set of atoms.

> ***"This program implements a hybrid of the fast multipole algorithm . . . and Barnes and Hut treecode algorithms . . . , and it reduces the computational complexity of evaluating electrostatic interactions for all pairs of atoms from $O(N^2)$ to $O(N)$."***

3 Design

The computations involved in each time step of MD can be broken into several portions. First, the forces acting on each atom are computed according to the empirical force field that approximates intramolecular forces. The force field used by NAMD is the CHARMM force field, which includes 2-, 3-, and 4-body interactions, electrostatic interactions, and van der Waals interactions. Once these forces are computed, a numerical integration scheme is used to update the positions and velocities of all the atoms. The force calculations are the major time-consuming portion of the simulation, and it is this that requires the most attention in parallelization. The numerical integrator currently used by NAMD is the velocity Verlet method (Allen and Tildesley, 1987), which represents a minor fraction of the overall computation of the simulation. More elaborate integration methods, which allow longer time steps and simulations over longer time scales, are being explored (Skeel and Biesiadecki, 1994).

The design of NAMD provides the functionality described previously while trying to achieve the three major goals of high performance, scalability to very large parallel computers, and modularity. The following sections describe key features of the NAMD design and how they help achieve these goals.

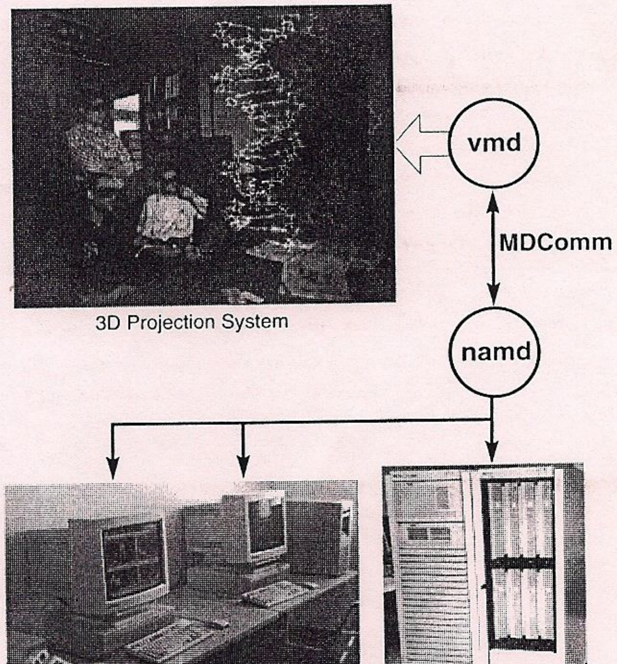

Fig. 1 Example of MDScope execution showing VMD running on a three-dimensional projection system and NAMD running on a cluster of HP workstations.

3.1 MULTIPLE TIME-STEP INTEGRATION WITH DPMTA

To further reduce the computational cost of computing full electrostatics, NAMD uses a multiple time stepping integration scheme. In this scheme, the total force acting

Table 1
Comparisons of NAMD Runtimes Using an 8-Å Electrostatic Cutoff and Using an 8-Å Local Interaction Length and DPMTA

System Name	Number of Atoms	Runtime with 8-Å Electrostatic Cutoff	Runtime with DPMTA
Polio virus coat	17,638	5,247	8,010
Estrogen receptor	10,238	3,386	5,089
Bacteriorhodopsin	3,762	1,026	1,411

NOTE: All the simulations were performed on four HP 735/125 workstations connected via ATM. The simulations consisted of 1,000 time steps. All runtimes are reported in seconds.

on each atom is broken into two pieces: a local component and a long-range component. The local force component consists of all bonded interactions as well as all nonbonded interactions for pairs that are separated by less than *local interaction length*. The long-range component consists only of electrostatic interactions outside of the local interaction length. Since the long-range forces are slowly varying, they are not evaluated every time step. Instead, they are evaluated every k time steps (Allen and Tildesley, 1987, p. 152), where each set of k time steps is referred to as a *cycle*. In the time steps between long-range force evaluations, the force from the previous evaluation is reused. For appropriate values of k, it is believed that the error introduced by this infrequent evaluation is modest compared to the error incurred by the use of the numerical (Verlet) integrator. As shown in Table 1, the performance of NAMD with the use of DPMTA to provide full electrostatics adds approximately 50% to the runtime of a simulation as compared to NAMD, which has an 8-Å electrostatic cutoff. Improved methods for incorporating the long-range forces, which reduce the frequency at which they need to be evaluated, have recently been implemented.

In the scheme described above, the van der Waals forces are neglected beyond the local interaction distance. Thus the van der Waals cutoff distance forms a lower limit to the local interaction distance. While this is believed to be sufficient, there are investigations under way to remove this limitation and provide full van der Waals calculations in $O(N)$ time as well.

3.2 SPATIAL DECOMPOSITION

A critical decision in the design of a parallel program is how to divide the work among processors so that equal amounts of work are assigned to each processor and so that the amount of memory and communication needed by each processor remains scalable. To define ideal scalability, consider a system of N atoms that requires time t to run on P processors, and uses memory m and communication bandwidth c on each node. If ideal scalability was to be achieved, t, m, and c would remain unchanged if both N and P were increased by a constant factor. Most current parallel MD programs divide the work among processors using a form of force decomposition. Using this method, all computational interactions are distributed in an equitable manner to the processors. However, in the naive case with fully replicated coordinates, such a decomposition requires $O(N)$ storage and communication. A more efficient decomposition proposed by Plimpton and Hendrickson

(1994) reduces the memory and communication requirements to $O(N/\sqrt{P})$ yet does not avoid the limit on scalability inherent with these methods.

To avoid these limitations, NAMD uses a spatial decomposition. In such a scheme, the spatial domain of the problem is split into disjoint regions of space, and these are then assigned to processors. Each processor computes interactions for only those atoms in its region, stores information for only those atoms, and for local interactions communicates those coordinates to only those processors assigned to neighboring regions. Therefore, this scheme scales nearly as $O(N/P)$ in terms of computation, memory, and communication. It provides NAMD with a decomposition that will scale efficiently to large numbers of processors. This spatial decomposition is used only for the local interactions that are directly calculated. The efficient handling of long-range nonbonded interactions is left to DPMTA, which was also designed to scale efficiently to large numbers of processors.

If a single region of a spatial decomposition is mapped to each processor, load-balancing problems due to inhomogeneous densities of atoms (e.g., density differences between protein and water and edge effects) arise. A decomposition that distributes the work load evenly across processors is difficult to achieve, and the computational effort needed to determine such a distribution is often quite large. To avoid this problem, NAMD employs a uniform decomposition of space coupled with multiple threads of control, which is described in the next section. A well-balanced decomposition is achieved by dividing the spatial domain of the problem into uniform cubes referred to as *patches*. A sample decomposition for a small polypeptide is shown in Figure 2. The length of a side of a patch is slightly longer than the local interaction length. Thus each patch communicates only with its neighboring patches to evaluate local interactions. Such a cubic decomposition is easy to compute and is used as a framework for force computation. It is assumed that there are many more patches than processors. The mapping of patches to processors can then be adjusted to balance the computational load across processors. The issue of load balancing is described in greater detail in section 5.2.

"It is assumed that there are many more patches than processors. The mapping of patches to processors can then be adjusted to balance the computational load across processors."

Spatial decomposition into uniform cubes with multiple cubes per processor has been used for parallel molecular dynamics by Esselink and Hilbers (1992). They obtained excellent results on a transputer network, with timings that agree well with the formula $\alpha + \beta(N/P)$, where α and β are constants. Their results support our design choice of spatial decomposition.

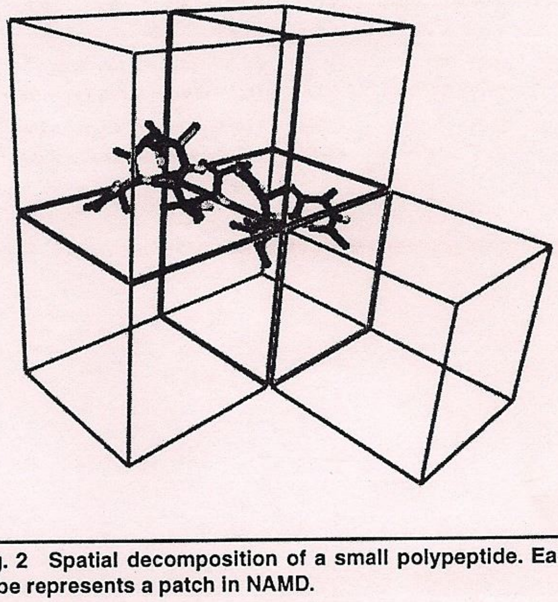

Fig. 2 Spatial decomposition of a small polypeptide. Each cube represents a patch in NAMD.

3.3 MULTIPLE THREADS OF CONTROL

NAMD uses a design with multiple threads of control, where each patch (described above) is implemented as an object that acts as its own thread of control. Each patch maintains its own state and contains functions that alter this state. Each patch is able to perform its own operations independent of the order of scheduling relative to other patches, or the processor that it is assigned to. In this manner, the same decomposition can be applied regardless of the number of processors that a simulation is running on.

By utilizing this scheme, NAMD maintains a simple spatial decomposition that is desirable for generality and ease of computation, while retaining the flexibility necessary to provide load balancing for irregular systems. The alternative, which is to use irregularly shaped regions of space that are constructed to provide computationally equal regions of space, would complicate communication patterns. Because of the separation between the computational responsibilities of a patch and the distribution of patches to processors, a variety of different schemes for distributing patches can be tried without changing any of the sequential code that performs the computations for the simulation.

3.4 MESSAGE-DRIVEN SCHEDULING

In order to execute the design that has been described, the scheduling of the computations to be performed should be done in a way that minimizes the idle time of each processor. In current parallel MD programs, the order in which computations are performed is fixed. If a computation requires information that has not yet arrived from another processor, the processor waits idle until this information arrives.

To avoid idle time, NAMD does not follow a set order in its computation. Instead, the order of computation is determined by the arrival of messages. In this way, a computation is scheduled only when all of the necessary data are available. This idea of message-driven scheduling (Kalé, 1990) is similar to that of active messages, which has received much attention recently (von Eicken et al., 1992). However, the message-driven scheduling implemented within NAMD is not interrupt driven as active messages are. Every computation in NAMD, including interactions requiring data only from the local processor, is scheduled by the receipt of a message. In addition, a priority scheme is used so that messages are processed in an efficient manner. Scheduling based on these priorities insures that messages that require return communications are processed before messages that require only computation. These ideas lead to a design that overlaps the computation time of one patch with the communication wait time of another patch.

To better understand how this message-driven scheduling combines with multiple threads of control to provide a design that is tolerant of communication latency, consider the simple example shown in Figure 3. In this example, three patches are maintained. Two of the patches, A and B, are assigned to processor 1, and patch C is assigned to processor 2. During each time step, each patch must perform the following tasks:

1. Send atom positions to neighboring patches
2. Receive atom positions from neighbors, calculate interactions between local atoms and neighbor atoms for which positions were received, and send the resulting forces
3. Calculate interactions between local atoms
4. Receive forces calculated by other patches and add these forces to the local forces
5. Perform a numerical integration step using these forces.

Assume that the patches communicate as shown, that is, patch C sends all of its atom positions to patch B and receives forces in return from B. Patch B sends all of its atom positions to patch A and receives forces in return. As shown in the two timing diagrams for processor 1, consider two scheduling algorithms. The first is a traditional scheduling method in which operations are performed in a fixed order, that is, both patches must perform each task before either continues to the next. The second is a message-driven algorithm. The fixed-order algorithm incurs idle processor time waiting for the interprocessor atoms position message to be sent from patch C to patch B. The message-driven algorithm overlaps this communication latency time with useful computations from patch A, which incurs no communication latency because all of its messages are intraprocessor in nature. While it is possible to determine a fixed-order schedule to overcome these problems for this simple example, doing so for a three-dimensional simulation with many processors and more complicated communication patterns is practically impossible. The automatic adaptiveness of message-driven scheduling makes it suitable for a wide range of processor and interprocessor network speeds. A comprehensive study of message-driven execution and its impact on performance can be found in (Gursoy, 1994).

3.5 OBJECT-ORIENTED DESIGN

Modularity is a major goal of NAMD, since it allows the program to be modified easily and allows multiple researchers to contribute efficiently to the program. In order to provide a high degree of modularity, NAMD is based on an object-oriented design. Using the object-oriented paradigm, the program consists of a small set of classes, each class consisting of objects of the same type of data structure and functions that can be performed on them. The level of modularity in such a design is very high, since the dependencies of one class on another are limited to the functions provided by the second class. The internal workings of each class are completely hidden from other classes. Thus the internal details of any class can be modified without affecting any other classes. Table 2 shows a summary of the classes present in NAMD and the functionality that each provides.

The most important class in NAMD is the `Patch`. A schematic diagram of the design of the `Patch` class is shown in Figure 4. Each `Patch` is responsible for maintaining the current position and velocity of every atom within its region (i.e., cube) of space. The `Patch` class is also responsible for calculating the forces acting on each of these atoms during each time step and using these forces to integrate the equations of motion to obtain new values for the position and velocity of each atom. Each `Patch` contains a set of force objects, each of which is responsible for computing a component of the force field. In addition, each `Patch` also contains an `Integrate` object that is responsible for using the current forces to perform numerical integration and determine new positions and velocities for atoms in its region of space. The majority of the logic in this class is used to coordinate the computation of forces and positions in a consistent manner. In addition, the procedures for transferring atoms from one region of space to another, as the simulation continues, is implemented by the `Patch` class.

To allow a flexible and modular implementation, each contribution to the force field (i.e., each distinct term in the energy potential function, such as 2-body bonds, 3-body bonds, harmonic boundary potentials, etc.) is implemented as a separate class. In this way, calculations of the force field components are completely separate. In addition, components of the force field can be enabled or disabled by creating or excluding specific objects. All of the force objects share a similar interface, which allows the `Patch` to utilize all of the force components in a very uniform manner. This interface includes functions for computing forces for local atoms, for computing interac-

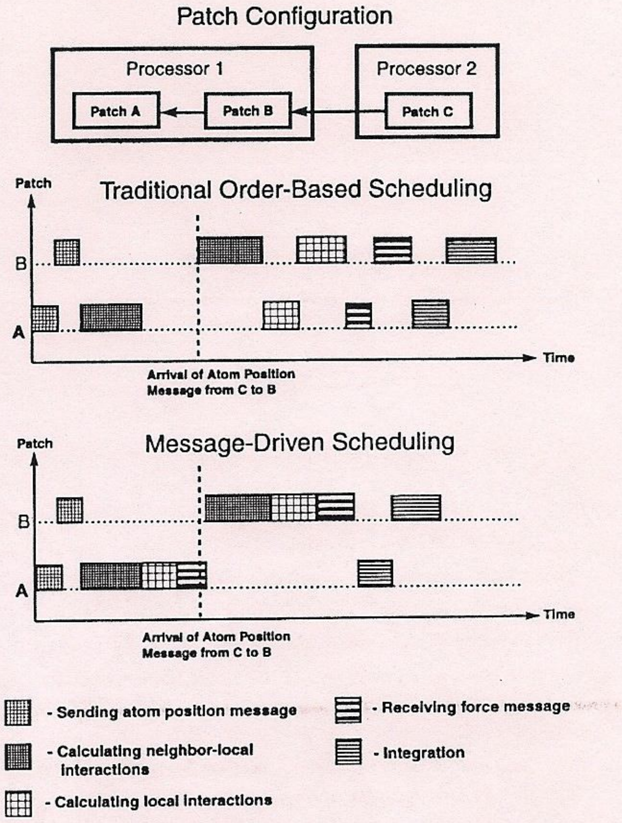

Fig. 3 Simple arrangement of threads along with timing diagrams showing how the tasks for processor 1 would be scheduled using a traditional order-based algorithm and using a message-driven algorithm. Note that the order-based algorithm incurs idle processor time while waiting for the atom position message from patch C to patch B to arrive, whereas the message-driven algorithm overlaps this message latency time with other computations.

Table 2
Description of the Purpose of the Objects Present on Each Processor in NAMD

Class Name	Description
`Communicate`	Protocol independent means of message passing, including operations such as send, receive, and broadcast
`Inform`	Means to print messages to the screen from any processor
`SimParameters`	Container class for static simulation data such as number of time steps, time-step size, and so on
`Molecule`	Container class for static structural data for the molecule such as which atoms are bonded by various types of bonds, explicit exclusions, and so on
`Parameters`	Container class for energy parameters from the parameter files
`LoadBalance`	Means for calculating an optimal distribution of patches to processors that will keep the load balanced across all processors
`Collect`	Collection mechanism for gathering global information such as energy totals
`Output`	Means of producing all forms of output for NAMD including trajectory files, energy output, and so on
`PatchDistrib`	Container class for the current processor assignment for all the patches in the simulation
`FMAInterface`	Processor-level interface to the full electrostatic module
`PatchList`	Container class for the patches belonging to this processor and the logic to schedule the execution of these patches

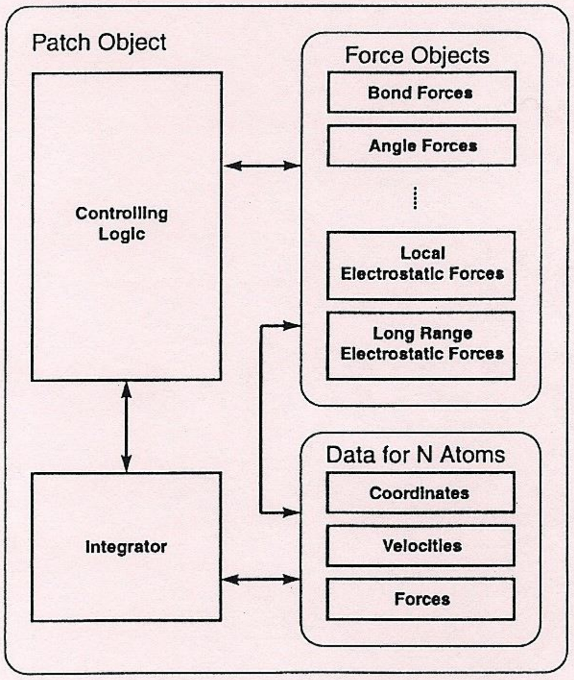

Fig. 4 Patch-level object diagram for NAMD. Each patch consists of a data section that includes the coordinates, velocities, and forces for each atom; force objects that compute components of the force field; an object to perform integration; and the controlling logic that calls each object appropriately.

tions between local atoms and atoms from neighboring patches, and for reporting energies.

4 Implementation

NAMD uses C++ to implement the design ideas described in the previous section in a modular and efficient manner. It is in this implementation that the critical ideas of the design are realized. The following subsections describe some of the important aspects of this implementation.

4.1 IMPLEMENTATION IN C++

The choice of a language with object-oriented constructs facilitates the implementation and enforcement of the object-oriented design. While implementation of these ideas in a traditional procedural language such as C is possible, it is not natural and can be easily bypassed.

Although C++ provides the object-oriented support that is desired, there are questions regarding its performance (Haney, 1994) and portability. To address these concerns, conscious efforts were made to avoid features that could adversely effect the performance or portability of the program. To avoid performance problems, the code that performs the actual numerical calculations is reduced to plain C code. In these portions of the code, complicated constructs that may not lead to optimal compilation are avoided. The code in-lining features of C++ are heavily used throughout NAMD. To avoid runtime function lookups, excessive use of virtual functions is avoided. In order to prevent possible portability problems, features of C++ that are not standard across platforms, such as templates, are avoided as well. By using these techniques, NAMD's implementation in C++ does not carry significant portability penalties over a similar implementation in a more traditional language such as C or FORTRAN. Performance comparisons between NAMD and other MD packages prove that this careful C++ implementation does, in fact, incur only minor performance penalties over implementations in FORTRAN and C.

> ***"Performance comparisons between NAMD and other MD packages prove that this careful C++ implementation does, in fact, incur only minor performance penalties over implementations in FORTRAN and C."***

4.2 OBJECT-ORIENTED INTERFACE TO DPMTA

An example of how object-oriented techniques are used to maintain the modularity of NAMD is the interface used to incorporate DPMTA. A key issue in the integration of the capabilities of NAMD and DPMTA was whether the two would share a common decomposition of the molecular system. In order to maintain the modularity and flexibility of both programs, an interface layer rather than a shared decomposition has been implemented. A schematic diagram of this interface is shown in Figure 5. The interface uses two classes: one that provides the interface that DPMTA expects and another that provides the interface that NAMD expects.

On the DPMTA side is the class `FMAInterface`. This class provides DPMTA with atom coordinates and a

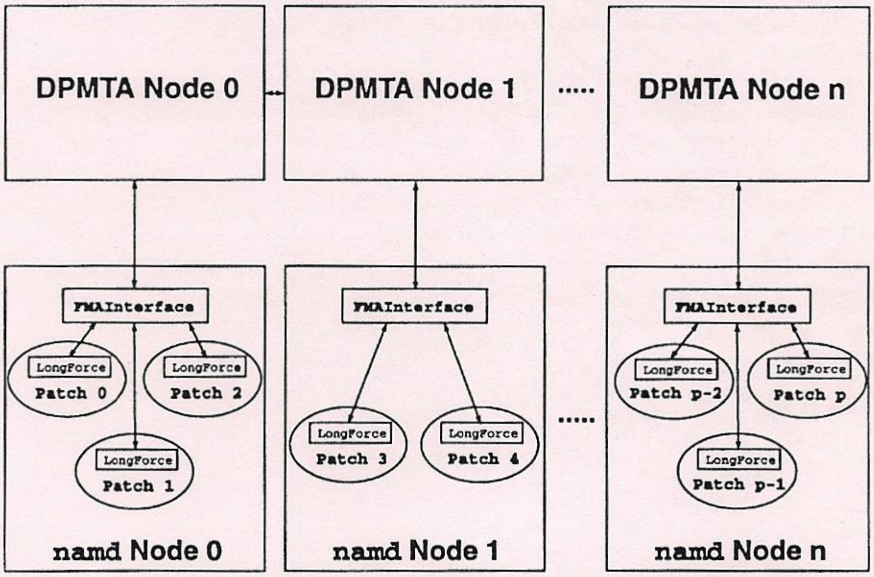

Fig. 5 Schematic diagram of the interface between DPMTA and NAMD. The interface consists of the classes `FMAInterface` and `LongForce`.

charge for every atom on the processor. On the NAMD side of the interface is the `LongForce` class. Its interface looks like the other force objects that reside in the `Patch` class. Each time that DPMTA is called to do force evaluation, each `LongForce` object passes the coordinates from its patch to the local `FMAInterface` object. When the `FMAInterface` object has received all of the coordinates for its processor, it invokes DPMTA. When the results are returned, the `FMAInterface` object is responsible for passing the results back to the appropriate `LongForce` objects.

This dual decomposition adds an insignificant amount of overhead. The interface maintains a very high level of modularity and abstraction between NAMD and DPMTA. As a demonstration of this modularity, these same objects were used to implement a direct calculation scheme for computing full electrostatics (for testing) without changing any source code outside of the interface classes.

4.3 PORTABLE PARALLELISM

It is intended that NAMD be portable across various parallel architectures, operating systems, and message-passing protocols. Several design features address this goal. The first is the complete separation of the parallel control and scheduling logic from the sequential computational portions of the code. This allows the parallel control structure of the program, which is relatively small, to change without affecting the sequential portions of the code. The second feature is the isolation of the majority of source code from protocol-specific function calls. Thus the amount of code that is dependent on the underlying protocol is small. The current release of NAMD uses these ideas to implement versions for different parallel systems, PVM (Geist et al., 1994) and Charm++ (Kalé and Krishnan, 1993).

NAMD using PVM. The PVM version of NAMD uses a `Communicate` object on each node to provide the send and receive protocols for the program. For messages that are sent between patches on the same node, this communication is handled entirely by the `Communicate` object on that node. For messages sent between nodes, the `Communicate` object on the sending node converts the message to a PVM packet and sends it to the receiving node using the PVM send routine. The `Communicate` object on the receiving node receives the message and converts to the form expected. While currently implemented only for PVM, the `Communicate` class could easily be adapted to any similar protocol providing simple sends and receives. A version of the `Communicate` class for the MPI communication protocol is being developed.

In addition to the `Communicate` class created for PVM, the message-driven scheduling for the processing of messages must be explicitly expressed, since PVM provides no such mechanism. This functionality is provided by the `PatchList` class. The major part of this class is simply a loop that polls the `Communicate` class for messages that have arrived and then schedules the execution of patches accordingly. This code also implements the priority scheme for the processing of messages.

NAMD using Charm++. Charm++ is a portable, object-oriented, message-driven parallel programming environment (Kalé and Krishnan, 1993). Charm++ is a superset of the C++ programming language that includes constructs for parallel implementations. In contrast to PVM, which provides only an underlying communication protocol, Charm++ includes an entire parallel system based on message-driven scheduling. Because of this, the logic that was created in the `PatchList` class for the PVM version is much more naturally and concisely expressed in the Charm++ version. Another significant advantage of Charm++ is the ability to maintain modularity without sacrificing message-driven scheduling. In the PVM version, any message that wishes to be handled in a message-driven fashion must be added to the control logic of the `PatchList` class explicitly. This creates dependencies and clutters the logic of the `PatchList`. In Charm++, the processing of all messages is scheduled in a message-driven fashion. A receive-based message-passing system cannot achieve this uniformity as elegantly or efficiently (Kalé and Gursoy, 1995).

In addition, the Charm++ programming environment includes several tools for analyzing the performance of parallel programs. One of these, Projections, provides visual output showing CPU utilization, message creation, message receipt, and other useful statistics. This tool has been applied to analyze the performance of NAMD.

5 Performance

The primary goal for NAMD is to provide the highest performance possible, thus allowing simulations of large systems faster than was previously possible. There are two important aspects to NAMD's performance. The first is its performance on a single node, and the second is the parallel performance, i.e., how efficiently the program is able to utilize parallel machines. The following sections discuss NAMD's current performance, how it compares to other MD packages, and plans for improving this performance in the future.

5.1 CURRENT PERFORMANCE

Although NAMD has only recently been released in its first version, and many performance improvements are still being made, it has already shown favorable results in comparison to other parallel molecular dynamics programs. Table 3 shows runtimes for NAMD, X-PLOR release 3.1, and CHARMM release c23f3 on a cluster of HP workstations connected via ATM, and Figure 6 shows a graph of the speedup versus the number of processors for these runs. The molecular system simulated is the protein calmodulin in a bath of 10,474 water molecules. The total number of atoms is 32,867, of which 32 are ions and 1413 are calmodulin united atoms. We measured the latency time for PVM on this hardware and software combination to be 419 microseconds per message and the bandwidth to be 85 Mbits/s. As these results show, NAMD is still slower than either program on a single node, which might be expected when comparing a sequential program that is relatively new to programs that have been optimized over a period of years. Improvements to the sequential computational portions of NAMD are currently being made that are expected to lessen the performance gap on a single node between NAMD and the other programs. However, NAMD appears to scale more efficiently than either of these programs as the number of nodes used for the simulation increases (up to the eight nodes used in this study). When eight processors are used, NAMD performs as well or better than either of the other programs. As larger numbers of processors are used, this performance difference is likely to be even more pronounced. The speedup of about six with eight processors for NAMD indicates the possibility of flattening of speedups with a larger number of processors; however, the speedups will be relatively high compared with the other programs. As a test that does not so conveniently lend itself to a parallel simulation, a 4,885-atom protein-DNA complex without water was simulated for 20 time steps. (This system was obtained from that described in section 6 by removing the water.) The execution time in seconds was 1,118, 655, 411, and 241 for one, two, four, and eight processors, respectively.

The current limitation on speedup can be attributed to two factors. First, on a cluster of workstations, the asynchronous interventions from the operating system impact performance, since some processors may slow the others down. Second, the load-balancing and the communication structure has not been optimized in the current implementation. Numerous efforts are under way to improve the parallel performance of NAMD. With these improve-

Table 3
Comparison of Runtimes for NAMD, X-PLOR, and CHARMM

Number of Processors	NAMD Runtime	X-PLOR Runtime	CHARMM Runtime
1	304.72	237.45	255.78
2	163.88	125.38	157.27
4	92.06	75.45	119.15
8	50.65	46.38	61.18

NOTE: All times are for 1,000-time-step simulations of a system of 32,867 atoms (calmodulin in 44-Å radius water sphere) using an 7.5-Å cutoff. All simulations were run on HP workstations connected via ATM. All runtimes are in minutes.

> *"However, NAMD appears to scale more efficiently than either of these programs as the number of nodes used for the simulation increases (up to the eight nodes used in this study)."*

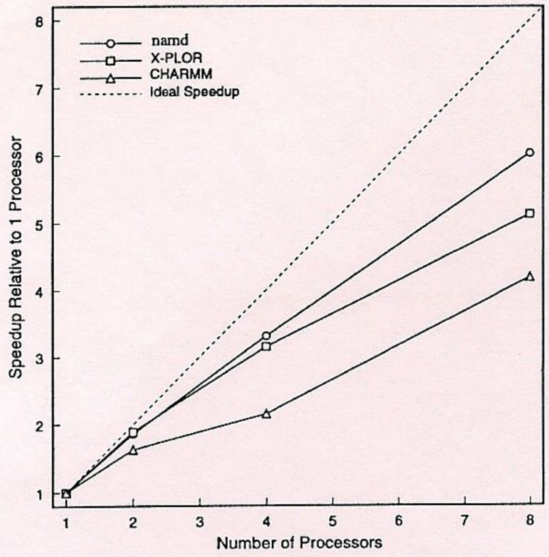

Fig. 6 Plot of speedup versus number of processors for NAMD, CHARMM, and X-PLOR. All simulations were for 1,000 time steps of a 32,867-atom system (calmodulin in a 44-Å radius water sphere), with a 7.5-Å cutoff run on a cluster of HP workstations.

ments, we expect to attain close to perfect speedups on stand-alone parallel machines with eight or more processors for the same-sized system. It is also important to note that speedups cannot continue to scale up to a large number of processors for the same—fixed size—molecular system, nor is that a goal of NAMD. Instead, we aim at scaled speedup; i.e., if the number of atoms (or more precisely, the total amount of computational work) is increased *k* times, we aim at finishing the computation in the same amount of time with *k* times more processors. Such scalable speedups have been demonstrated by Esselink and Hilbers (1992) using spatial decomposition in a relatively homogeneous context with periodic boundary conditions. In the near future, with further ongoing research, we expect to demonstrate scalable speedups for large inhomogeneous molecular systems, both for relatively regular domains generated by periodic boundary conditions and for the irregular domains that exist in simulations without periodicity.

One of the most important issues for the parallel performance of NAMD is load balancing, which is discussed in the next subsection. Other performance issues are discussed in the subsequent subsection.

5.2 LOAD BALANCING

The problem of load balancing within NAMD is that of finding an optimal distribution of patches to processors. The algorithm for doing this involves balancing the computational and communication costs for the patches assigned to each processor. An effective algorithm must try to preserve the locality of patches. All patches must communicate with neighboring patches, but communication with a patch on the same processor is inexpensive, requiring only the passing of a memory address from one object to another. Interprocessor communication is considerably more expensive, requiring data to be transferred across the network-connecting processors. Thus the load-balancing scheme must try to balance the computational load across processors while preserving the locality of patches.

The distribution method currently implemented within NAMD is based on recursive, orthogonal bisection (Berger and Bokhari, 1987). This distribution method relies on a heuristic cost function to approximate the load on each patch. This function factors in not only the number of atoms but also the number of patches per processor, local communication costs, and interprocessor communication costs. The distribution method tries to balance the sum of the cost function for all the patches assigned to each processor.

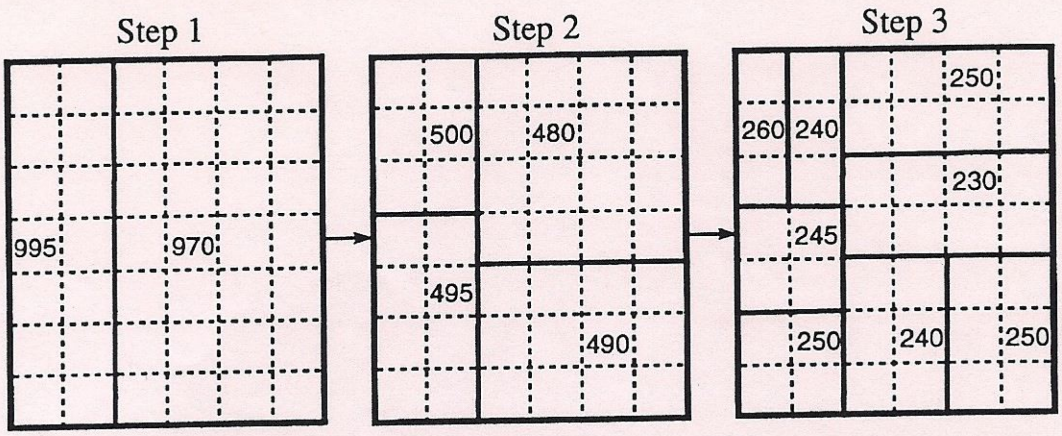

Fig. 7 Example of recursive bisection in two dimensions. Dashed lines represent patch boundaries; solid lines represent boundaries determined by recursive bisection. Numbers indicate cost function value for each bisection region.

Recursive bisection is used to partition the patches into rectangular sections that are assigned to processors. This procedure is demonstrated for two dimensions in Figure 7. It involves splitting the entire region into two portions, each of which has an approximately equal value for the heuristic cost function. Then, each of these two sections is subdivided in a similar manner. This procedure is repeated recursively until the system has been partitioned into as many sections as there are processors. This method is limited in its flexibility, however, since it can only divide the system into rectangular sections. Improvements in this algorithm are planned and are briefly discussed in the following subsection.

5.3 FUTURE LOAD-BALANCING ENHANCEMENTS

As mentioned previously, the current method for the initial distribution of patches to processors is limited in that it can divide the system only into rectangular regions. To remove this limitation, procedures to move small convex regions of patches from heavily loaded processors to less heavily loaded neighboring processors are being developed. By using smaller, more irregular shapes to assign patches to processors, a more accurate balancing of the cost function can be obtained. In addition, an even finer tuning of the balance of the cost function is being implemented. For each pair of neighboring patches, Newton's third law is exploited to reduce the cost of computing local electrostatics. In this method, one patch sends all of its atoms coordinates to the neighboring patch. The receiving patch then computes electrostatic interactions between the two sets of atoms and returns the forces to the original patch. Thus for this pair of patches, only one bears the computational cost for the electrostatic computations between them. In this procedure, the assignment of the electrostatic computation to pairs of patches that reside on separate processors is examined and adjusted to balance the load between processors (Windemuth, 1995).

Another concern is that of maintaining load balance as a simulation progresses. Even if the initial distribution provides an even balance across all processors, the system may become imbalanced as atoms move during the course of the simulation. To address this issue, NAMD currently gathers detailed statistics concerning the computation and idle times of all patches and processors involved in the simulation. Various schemes are being explored to use this information to maintain the load balance as the simulation continues. One scheme involves infrequently using these statistics along with the initial patch distribution scheme to remap patches to processors. Another is to perform a frequent incremental patch reassignment procedure involving small changes.

5.4 OTHER OPTIMIZATIONS

Other optimizations implemented in NAMD range from better algorithms and methods for single-node computation to better messaging schemes to enhance the parallel performance and scalability of NAMD. Some of these optimizations will be described in this subsection.

A first optimization concerns the elimination of duplicate messages. Each patch sends the coordinates of all of its atoms to a subset of its neighbors. These messages are the largest messages sent in NAMD, and identical information is sent to each patch. In a naive implementation, if a patch has three neighboring patches that reside on another processor, it would send three copies of the same message to this processor. To eliminate this redundancy, a single message is sent and delivered to all three patches instead of sending separate messages to each patch. This is done in such a way that no patch is aware that it is processing a message that will be or was used by other patches, thereby minimizing the code changes necessary for this optimization.

A further optimization is for communication networks such as ATM, where a high bandwidth is provided but with a fairly high latency cost per message. With such a network, sending a few larger messages is more efficient than sending many smaller messages, since the message latency is incurred fewer times. To accomplish this, the `Communicate` class combines multiple messages to be sent to the same processor into one physical message to be sent over the network. This device is used in situations such as the beginning of a time step, when many messages are sent at the same time to the same processor. It cannot be used for all messages destined for some given node without incurring significant delay. This optimization has provided a significant performance increase on a network of workstations connected via ATM.

6 Application to a Large Supramolecular System

In recent years the structures of rather large biomolecules, encompassing 10,000 to 100,000 atoms, have been determined. Examples are the photosynthetic reaction center (Deisenhofer et al., 1985; Feher, Arno, and Okamura, 1988), the myosin head group (Rayment et al., 1993), ATPase (Abrahams et al., 1994), and cytochrome oxydase (Iwata et al., 1995). Furthermore, biomolecular systems consisting of aggregates of molecules have recently become targets of simulation studies, for example, lipid bilayers (Pastor and Venable, 1993; Heller, Schaefer, and Schulten, 1993; Zhou and Schulten, 1995), membrane-protein complexes (Zhou and Schulten, 1996), F-actin (Lorenz, Popp, and Holmes, 1993; Tirion et al., 1995), and protein-DNA complexes (Beveridge and Ravishanker, 1994; Eriksson, Härd, and Nilsson, 1995; Harris et al., 1994). The size of these systems, often requiring immersion in a bath of water, exceeds by about an order of magnitude the size of systems conventionally simulated today, that is, about 5,000 atoms. Experimental molecular biology is targeting larger and larger structures, in particular, supramolecular structures, and the field of computational structural biology needs to advance its tools to follow this trend. The development of a simulation program like NAMD is particularly timely in this respect.

As an example, NAMD has been applied to simulate a complex of a protein with a segment of DNA in a bath of salt water, comprising altogether 36,000 atoms. The protein chosen is the estrogen receptor (ER), a member of the steroid hormone receptor family. This protein functions as a ligand-dependent transcription factor and contains three main functional domains: the NH_2-terminal ligand-binding domain, which affects transcription efficiency; the central DNA-binding domain, which binds to the target gene, that is, it recognizes a specific DNA sequence; and the COOH-terminal hormone-binding domain, which binds to estrogen. At present, only the structure of the DNA-binding domain is known (Schwabe et al., 1993). The estrogen receptor is known to function as a dimer in binding to DNA and regulating transcription. Accordingly, our sample system includes two estrogen receptors but only their DNA-binding domains. The simulation is similar to that of a complex of DNA with a dimer of a glucocorticoid receptor reported in Bishop and Schulten (1994, 1996).

The crystallographic structure of the estrogen receptor reported in Schwabe et al. (1993) contains two units, each with a DNA-binding domain dimer, the response element, and water molecules, but one unit is not completely resolved. A complete unit of this structure was used to prepare an initial model of the ER-DNA-water system, which is composed of a DBD dimer, the ER response element, and ions embedded in a sphere of 11,000 water molecules. The volume of water is large enough to not only incorporate the whole system but also to allow the DNA enough flexibility to unwind and bend.

Molecular dynamics simulations were performed on this system with NAMD using a cluster of HP workstations connected via an ATM switch. Using eight workstations, 50 ps of simulation time can be computed in a matter of 2 or 3 days.

Because of the highly charged nature of the system, it is also an ideal case for the use of full electrostatics using DPMTA. This is a feature unique to NAMD. The simulations, initially, will identify which protein-DNA and protein-protein interactions are involved in recognition of the DNA sequence by the protein and which interactions and degrees of freedom are involved in the bending and

> *"As an example, NAMD has been applied to simulate a complex of a protein with a segment of DNA in a bath of salt water, comprising altogether 36,000 atoms."*

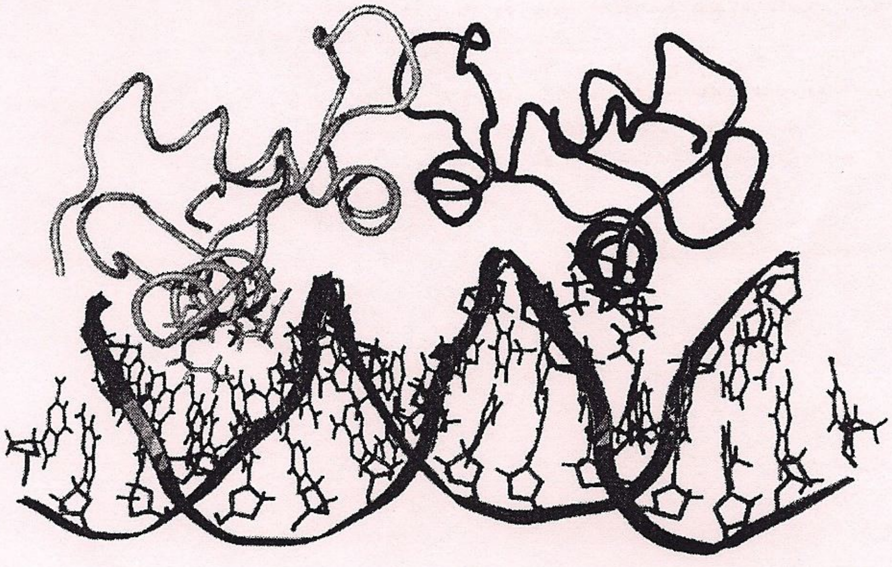

Fig. 8 **A dimer of the ER-DBD complexed with DNA, drawn using the graphics program VMD.**

unwinding of the DNA, which is expected to occur on protein binding. The estrogen receptor system being simulated is shown in Figure 8 in a rendering that used the graphics program VMD.

7 Conclusions

The program NAMD is an inherently parallel program for fast simulations of large biomolecular systems. It is designed to efficiently utilize parallel machines ranging in size from tens of processors to hundreds of processors. For this purpose, the program uses a spatial decomposition of the molecular system combined with a multithreaded, message-driven design to provide a potentially highly scalable design that is tolerant of communication latency. An object-oriented design and implementation make NAMD highly modular. This renders NAMD an excellent testbed for experimentation with new algorithms. One new algorithm that has already been incorporated is DPMTA, which provides an $O(N)$ means of computing electrostatic interactions for all pairs of atoms. In future work, we plan to implement new load-balancing algorithms and communication optimizations, experiment with alternate parallelization strategies, and carry out performance-tuning exercises, leading to an improved program that will efficiently run very large simulations (of systems beyond 100,000 atoms) using hundreds of processors. As part of the MDScope package, NAMD provides the computational engine for an interactive modeling system. The program is currently being used for the study of several molecular systems such as the estrogen receptor. With its parallel design and modularity, it is hoped that NAMD will be an excellent MD program to explore new algorithms and to perform new simulations that were previously impossible.

8 Availability

Information concerning NAMD and MDScope development can be found via the World Wide Web at the address `http://www.ks.uiuc.edu`. Source code and documentation for NAMD and the other components of MDScope are available via anonymous ftp from the site `ftp.ks.uiuc.edu` in the directory `/pub/mdscope`.

ACKNOWLEDGMENTS

The authors would like to thank the members of the Theoretical Biophysics group at the University of Illinois and the Beckman Institute for many useful suggestions and willing help in testing NAMD, with a special thanks to Dorina Kosztin for the information and figure concerning the estrogen receptor, to Willy Wriggers for his help in running the X-PLOR and CHARM performance tests, and to Robert Brunner for running additional performance tests. The authors gratefully acknowledge the support of grants from the National Institutes of Health (PHS 5 P41 RR05969-04), the National Science Foundation (BIR-9318159), and the Roy J. Carver Charitable Trust.

BIOGRAPHIES

Mark T. Nelson received a B.S. degree in general engineering from the University of Illinois at Urbana-Champaign in 1991. He then joined AT&T Network Systems, where he developed software for E9-1-1 systems. In 1995 he earned an M.S. degree in computer science from the University of Illinois at Urbana-Champaign. He is currently pursuing a Ph.D. in computer science in the Theoretical Biophysics group.

William F. Humphrey received a B.S. degree in physics, math, and computer science from Drake University in 1990 and an M.S. degree in physics from the University of Illinois at Urbana-Champaign in 1992. He is currently near completion of a Ph.D. in physics in the Theoretical Biophysics group.

Attila Gursoy received a B.S. degree in computer science from Middle East Technical University, Ankara, Turkey in 1986 and an M.S. in computer science from Bilkent University, Ankara. In June 1988 he received an M.S. in computer science. He continued his research in parallel programming environments, earning a Ph.D. in computer science from the University of Illinois at Urbana-Champaign in 1994. Currently, he is a postdoc in the Theoretical Biophysics group.

Andrew Dalke received B.S. degrees in math, physics, and computer science from Florida State University in 1992 and an M.S. degree in physics from the University of Illinois at Urbana-Champaign in 1993. He is currently on leave from the UIUC physics Ph.D. program and is working as a research programmer for the Theoretical Biophysics group.

Laxmikant V. Kalé received a B.Tech. degree in electronics engineering from Benares Hindu University, Varanasi, India, an M.E. in computer science from the Indian Institute of Science in Bangalore, and a Ph.D. in computer science from the State University of New York, Stony Brook. He spent 2 years at the Tata Institute of Fundamental Research. In 1985 he went to the University of Illinois at Urbana-Champaign, where he is now associate professor of computer science. His research interests include parallel programming tools, performance feedback and debugging, scientific and engineering applications of large-scale parallel computing, and parallel processing for AI. He is a member of the IEEE, IEEE Computer Society, and ACM.

Robert D. Skeel is professor of computer science at the University of Illinois at Urbana-Champaign, where he has been since 1973. His research interest is in computational methods for biomolecular modeling. He received a B.S. in applied mathematics from the University of Alberta, an M.S. in mathematics from the University of Toronto, and a Ph.D. in computing science again from Alberta. He is a member of SIAM and is on the editorial board for *SIAM Journal on Scientific Computing*.

Klaus Schulten received a Ph.D. in chemical physics from Harvard University in 1974. He then spent 6 years as a research assistant at the Max Planck Institute for Biophysical Chemistry in Göttingen, Germany, and 8 years as associate professor of physics at the Technical University of Munich. Since 1988 he has been professor of physics, chemistry, biophysics, and electrical and computer engineering at the University of Illinois at Urbana-Champaign and leader of the Theoretical Biophysics group at the Beckman Institute in Illinois. His research interests focus on theoretical and computational physics and biology.

REFERENCES

Abrahams, J. P., Leslie, A. G. W., Lutter, R., and Walker, J. E. 1994. Structure at 2.8 angstoms resolution of F_1-ATPase from bovine heart mitochondria. *Nature* 370:621-628.

Allen, M. P., and Tildesley, D. J. 1987. *Computer simulation of liquids*. New York: Oxford University Press.

Barnes, J., and Hut, P. 1986. A hierarchical $\mathcal{O}(N\log N)$ force-calculation algorithm. *Nature* 324:446-449.

Berger, M. J., and Bokhari, S. H. 1987. A partitioning strategy for nonuniform problems on multiprocessors. *IEEE Transactions on Computers* C-36:570-580.

Beveridge, D. L., and Ravishanker, G. 1994. Molecular dynamics studies of DNA. *Curr. Opinion Struct. Biol.* 4:246-255.

Bishop, T., and Schulten, K. 1994. Molecular dynamics study of a sequence specific protein-DNA interaction. In *Computational approaches in supramolecular chemistry*, edited by G. Wipff. Boston: Kluwer Academic, pp. 419-439.

Bishop, T., and Schulten, K. 1996. Molecular dynamics study of glucocorticoid receptor-DNA binding. *Proteins: Structure, Function, and Genetics* 24(1):115-133.

Brooks, B. R., Bruccoleri, R. E., Olafson, B. D., States, D. J., Swaminathan, S., and Karplus, M. 1983. CHARM: A program for macromolecular energy, minimization, and dynamics calculations. *J. Comp. Chem.* 4:187-217.

Brooks, B. R., and Hodošček, M. 1992. Parallelization of CHARM for MIMD machines. *Chemical Design Automation News* 7:16-22.

Brooks, C. L. III, and Karplus, M. 1983. Deformable stochastic boundaries in molecular dynamics. *J. Chem. Phys.* 79(12):6312-6325.

Brooks, C. L. III, Karplus, M., and Pettitt, B. M. 1988. *Proteins: A theoretical perspective of dynamics, structure and thermodynamics*. New York: John Wiley & Sons.

Brünger, A. T. 1992. X-PLOR, version 3.1: A system for X-ray crystallography and NMR. The Howard Hughes Medical Institute and Department of Molecular Biophysics and Biochemistry, Yale University.

Carson, M., and Hermans, J. 1985. The molecular dynamics workshop laboratory. In *Molecular dynamics and protein structure*, edited by J. Hermans. Chapel Hill: University of North Carolina Press.

Clark, T., Hanxleden, R., McCammon, J., and Scott, L. 1994. Parallelizing molecular dynamics using spatial decomposition. In *Proceedings of the Scalable High Performance Computing Conference, Knoxville, TN*. Los Alamitos, CA: IEEE Computer Society Press, pp. 95-102.

Deisenhofer, J., Epp, O., Mikki, K., Huber, R., and Michel, H. 1985. Structure of the protein subunits in the photosynthetic reaction centre of *Rhodopseudomonas viridis* at 3 Å resolution. *Nature* 318:618-624.

Eriksson, M., Härd, T., and Nilsson, L. 1995. Molecular dynamics simulations of the glucocorticoid receptor DNA-binding domain in complex with DNA and free in solution. *Biophys. J.* 68:402-426.

Esselink, K., and Hilbers, P. 1992. Parallel molecular dynamics on a torus network. In *Proc. Scalable High Peformance Computing*, Williamsburgh, VA. Los Alamitos, CA: IEEE Computer Society Press, pp. 106-112.

Feher, G., Arno, T. R., and Okamura, M. Y. 1988. The effect of an electric field on the charge recombination rate of $D^{+}\mathrm{Q}_{A}^{-} \rightarrow DQ_{A}$ in reaction centers from *rhodobacter sphaeroides* R-26. In *The photosynthetic bacterial reaction center: Structure and dynamics*, edited by J. Breton and A. Vermeglio. New York and London: Plenum, pp. 271-287.

Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R., and Sunderam, V. 1994. PVM 3 users guide and reference manual. Technical Manual ORNL/TM-12187. Oak Ridge National Laboratory, TN.

Greengard, L., and Rokhlin, V. 1987. A fast algorithm for particle simulation. *J. Comp. Phys.* 73:325-348.

Gursoy, A. 1994. Simplified expression of message-driven programs and quantification of their impact on performance. Ph.D. thesis, University of Illinois at Urbana-Champaign.

Haney, S. W. 1994. Is C++ fast enough for scientific computing? *Computers in Physics* 8:690-694.

Harris, L. F., Sullivan, M. R., Popken-Harris, P. D., and Hickok, D. F. 1994. Molecular dynamics simulations in solvent of the glucocorticoid receptor protein in complex with a glucocorticoid response element DNA sequence. *J. Biom. Struct. Dyn.* 12:249-270.

Heller, H., Schaefer, M., and Schulten, K. 1993. Molecular dynamics simulation of a bilayer of 200 lipids in the gel and in the liquid crystal-phases. *J. Phys. Chem.* 97:8343-8360.

Iwata, S., Ostermeier, C., Ludwig, B. and Michel, H. 1995. Structure at 2.8 *A* resolution of cytochrom *C* oxidase from *Paracoccus denitrificans*. *IWAT95* 376:660-669.

Kalé, L. V. 1990. The Chare Kernel parallel programming language and system. In *Proc. Int. Conf. on Parallel Processing*, vol. 2, pp. 17-25. Boston: CRC Press.

Kalé, L. V., and Gursoy, A. 1995. Modularity, reuse and efficiency with message-driven libraries. In *Proc. 7th SIAM*

Parallel Processing Conf., San Francisco, pp. 738-743. Philadephia: SIAM.

Kalé, L. V., and Gursoy, A. 1995. Modularity, reuse and efficiency with message-driven libraries. In *Proceedings of the 7th SIAM*.

Kalé, L. V., and Krishnan, S. 1993. CHARM++: A portable concurrent object-oriented system based on C++. In *Proc. OOPSLA-93*, Washington, DC. New York: ACM Press.

Lorenz, M., Popp, D., and Holmes, K. C. 1993. Refinement of the F-actin model against X-ray fiber diffraction data by the use of a directed mutation algorithm. *J. Mol. Biol.* 234: 826-836.

McCammon, J. A., and Harvey, S. C. 1987. *Dynamics ofproteins and nucleic acids*. Cambridge, UK: Cambridge University Press.

Nelson, M., Humphrey, W., Gursoy, A., Dalke, A., Kalé, L., Skeel, R., Schulten, K., and Kufrin, R. 1995. MDScope—A visual computing environment for structural biology. *Comput. Phys. Commun.* 91(1-3):111-134.

Pastor, R. W., and Venable, R. M. 1993. Molecular and stochastic dynamics simulation of lipid molecules. In *Computer simulation of biomolecular systems: Theoretical and experimental applications*, edited by W. F. van Gunsteren, P. K. Weiner, and A. K. Wilkinson. Leiden: ESCOM Science Publishers, pp. 443-463.

Plimpton, S., and Hendrickson, B. 1994. A new parallel method for molecular dynamics simulation of macromolecular systems. Technical Report SAND94-1862. Sandia National Laboratories, NM.

Rankin, W., and Board, J. A., Jr. 1995. A portable distributed implementation of the parallel multipole tree algorithm. Duke University Technical Report 95-002. *IEEE Symposium on High Performance Distributed Computing* (pp. 17-22). Los Alamitos, CA: IEEE Computer Society Press.

Rayment, I., Rypniewski, W. R., Schmidt-Bäse, K., Smith, R., Tomchick, D. R., Benning, M. M., Winkelmann, D. A., Wesenberg, G., and Holden, H. M. 1993. Three-dimensional structure of myosin subfragment-1: A molecular motor. *Science* 261:50-58.

Schwabe, J. W. R., Chapman, L., Finch, J. T., and Rhodes, D. 1993. The crystal structure of the estrogen receptor DNA-binding domain bound to DNA: How receptors discriminate between their response elements. *Cell* 75:567-578.

Skeel, R. D., and Biesiadecki, J. J. 1994. Symplectic integration with variable stepsize. *Annals Numer. Math.* 1:191-198.

Tirion, M. M., ben-Avraham, D., Lorenz, M., and Holmes, K. C. 1995. Normal modes as refinement parameters for the F-actin model. *Biophys. J.* 68:5-12.

van Gunsteren, W. F., and Berendsen, H. J. C. 1987. *GROMOS manual*. BIOMOS b. v., Laboratory of Physical Chemistry, University of Groningen.

von Eicken, T., Culler, D. E., Goldstein, S. C., and Schauser, K. E. 1992. Active messages: A mechanism for integrated communication and computation. In *Proc. Int. Symp. on Computer Architecture*. New York: ACM Press.

Weiner, P. K., and Kollman, P. A. 1981. AMBER: Assisted model building with energy refinement: A general program for modeling molecules and their interactions. *J. Comp. Chem.* 2:287.

Windemuth, A. 1995. Advanced algorithms for molecular dynamics simulation: The program PMD. In *Parallel computing in computational chemistry*, edited by T. Mattson. Washington, DC: ACS Books.

Zhou, F., and Schulten, K. 1995. Molecular dynamics study of a membrane-water interface. *J. Phys. Chem.* 99:2194-2208.

Zhou, F., and Schulten, K. 1996. Molecular dynamics study of the activation of phospholipase A_2 on a membrane surface. *Proteins: Structure, Function, and Genetics* 25(1):12-27.