MDScope - A Visual Computing Environment for Structural Biology

Mark Nelson, William Humphrey, Attila Gursoy, Andrew Dalke

Laxmikant Kale, Robert Skeel, Klaus Schulten*

Theoretical Biophysics Group
University of Illinois and Beckman Institute
405 North Matthews
Urbana, IL 61801

Richard Kufrin

National Center for Supercomputing Applications
605 East Springfield Avenue
Champaign, IL 61820

Classification: Atomic and Molecular Dynamics

January 6, 1995

* To whom correspondence should be addressed. Email: kschulte@ks.uiuc.edu

Abstract

MDScope is an integrated set of computational tools which function as an interactive visual computing environment for the simulation and study of biopolymers. This environment consists of three parts: (1) vmd, a molecular visualization program for interactive display of molecular systems; (2) namd, a molecular dynamics program designed for performance, scalability, modularity, and portability, which runs in parallel on a variety of computer platforms; (3) MDComm, a protocol and library which functions as the unifying communication agent between the visualization and simulation components of MDScope. namd is expressly designed for distributed memory parallel architectures and uses a spatial decomposition parallelization strategy coupled with a multi-threaded, message-driven computation model which reduces inefficiencies due to communication latency. Through the MDComm software, vmd acts as a graphical interface and interactive control for namd, allowing a user running namd to utilize a parallel platform for computational power while visualizing the trajectory as it is computed. Modularity in both vmd and namd is accomplished through an object-oriented design, which facilitates the addition of features and new algorithms.

1 Introduction

Molecular dynamics (MD) simulations [24, 7] are playing an increasingly important role for the study of structure and function of biomolecular systems and in drug design. MD simulations have been applied to the refinement of structures derived from X-ray diffraction and NMR data, to the investigation of enzymatic mechanisms and enzyme inhibitors, to the study of biopolymer aggregates like membranes, and to protein structure prediction. Widely used MD applications include CHARMm [5], X-PLOR [8], GROMOS [29], AMBER [30], and CEDAR [10]. Unfortunately, MD simulations are computer time-intensive and, currently, simulations require several hours or days to complete.

Were it possible that MD simulations could be realized interactively, exciting applications such as structure refinement or structure prediction in drug design would become feasible since simulations could involve direct feedback from users. For this purpose, a researcher would need to have immediate access to the results of MD simulations through a suitable molecular graphics user interface. Such interaction would require a certain level of computational and display performance, i.e., 100 fs of motion for a protein with 1000 atoms would need to be computed and visualized within 10 s. In this article we describe the development of a *visual computing environment* for interactive simulation of biopolymers, a software system called MDScope, which takes an important step toward this goal.

The bottleneck for interactive MD is the speed of processors carrying out MD simulations. Available computer graphics systems as well as available network technology can sustain the requirements for the transfer of MD trajectories to a graphics computer and the subsequent image rendering. The processor performance bottleneck can be overcome, however, by using

the peak capabilities of parallel computing platforms such as dedicated large-scale parallel computers or clusters of high-performance workstations. This requires a molecular dynamics program which runs efficiently on such platforms and is well integrated with a molecular graphics application.

MDScope provides such a system. MDScope combines (1) a molecular visualization program vmd, (2) an MD simulation program namd designed for distributed memory parallel computers, and (3) a communication protocol and library MDComm which allows vmd to function as a graphical interface to namd, unifying these applications into a single visual computational environment. Combined with a large screen stereo projection device, as shown in fig. 1, MDScope provides an affordable tool for the study of biopolymer structures. All components of MDScope, including the complete source code, are available free of charge.

Figure 1 here

namd has been developed from the beginning for parallel computer architectures and with the goal of efficiency and scalability. Key design elements of namd are a spatial decomposition strategy to partition the parallel computation tasks, as well as a multi-threaded, message-driven method of execution. This design uses a number of independent computation tasks for each processor in a parallel computer, instead of one single task; control is transferred between threads based on availability of messages from other processors, which reduces inefficiency due to communication latency. namd is written in C++, an object-oriented extension of the C programming language, and includes a descriptive programmers guide as well as extensive source code documentation.

vmd, the visualization component of MDScope, allows the user to interactively display and control any number of remote MD simulations running simultaneously on remote su-

percomputers or high-performance workstations. vmd is designed to support a wide variety of display and input devices, e.g., stereo display projectors as shown in fig. 1 and spatial tracking devices which may provide a three-dimensional pointer.

In this paper we describe the individual components of MDScope: vmd (section 2), namd (section 3), and the MDComm software (section 4). These components are each capable of functioning separately in their stated roles; we discuss the use of these components as separate applications, and as an integrated computational environment. In section 5 an example of the use of this system, the simulation of components of the coat of poliovirus, is presented. Information on the availability and location of MDScope software is given in section 6.

2 The Visualization Program vmd

vmd is a molecular visualization program developed for interactive display and manipulation of biopolymer structures. It provides the user with both a textual and graphical user interface, and is designed to be used concurrently with MD simulation programs, such as the program namd described in section 3. vmd has three objectives, the visualization of molecular dynamics, the control of remote MD simulations, and a modular, easily modifiable design.

Dynamics Visualization A fundamental purpose of vmd is to visualize structures as well as dynamics of macromolecular systems, in particular, biopolymers such as proteins, nucleic acids, and membranes. Several excellent packages exist for this purpose, both commercial (e.g., Quanta [27]), and non-commercial (e.g., Ribbons [9], XMol [25], Midas Plus [13], and

others), for a variety of computer platforms. vmd contains many of the standard molecular visualization options of existing programs, but focuses on visualization of the *dynamics* of molecules. vmd furnishes interactive graphical visualization of biopolymer systems, independent of the source of the dynamically varying molecular quantities (position, velocity, energy); the latter may be obtained either from completed MD calculations or from concurrently running MD simulations.

Several different display and input devices are available for interactive visualization, such as head-mounted displays or stereo monitors, and an objective of vmd is to support the use of such devices. A particular objective is the use of large-screen projectors capable of displaying stereo images, coupled with hand-held spatial tracking devices conveying position and orientation for graphical object manipulation; this system offers an excellent collaborative environment for the study of molecular systems, where several researchers can simultaneously view and manipulate three-dimensional representations of biopolymer structures.

Remote Simulation Control A major difference between vmd and several of the current molecular visualization applications is the capability for direct interaction and control of MD simulations running simultaneously on a remote workstation or supercomputer. vmd connects to a remote computer, and either attaches to a currently running job or begins a new simulation. vmd itself is independent of the specific MD application selected. Through the use of the MDComm software (section 4), which provides a defined protocol for communication between a local visualization program and a remote MD program, vmd is capable of obtaining and displaying the state of any number of different MD applications which also

use MDComm. Multiple remote simulations can be attached to vmd simultaneously, and from vmd the state of each simulation can be monitored and interactively controlled.

Modularity vmd is written in C++, using an object-oriented methodology, and is well suited for rapid extensions and additions such as support for new molecular file formats, display and input devices, and graphics rendering libraries. Major targets for this program are research groups working in structure refinement, structure prediction, drug design, mechanisms of biopolymers, and aggregates of biopolymers. These groups may be either interested in using the current implementation of vmd, modifying the existing program structure, or adding required features not currently present. The use of an object-oriented design greatly speeds up and improves the capability for modification and enhancement of the program. A comprehensive programmers guide as well as a users guide describe the structure and design of the objects making up the program, as well as the overall layout of the program.

2.1 Features

Key features of vmd include:

- molecular rendering and coloring options for simultaneous display of any number of molecules;
- atom selection syntax for choosing atom subsets for display and rendering characteristics;
- images displayed in stereo, using a side-by-side format, or Crystal-Eyes stereo mode for suitably equipped systems;

- simultaneous connection to remote MD simulations on any remote UNIX workstation or supercomputer;
- support for use of spatial tracking devices which function as a 3D pointer, with accompanying 3D user interface in a stereo display environment;
- control of all actions via input scripts as well as through the graphical user interface;
- session logging and playback for all program capabilities.

2.2 Design Overview

vmd employs an *object-oriented* design, which endows the program with a modular structure. The object-oriented design is based on individual software components which contain data and methods for manipulating the corresponding data. This design also promotes the concept of *inheritance* and *polymorphism*, which allow a software object to add functionality to a more general base or parent object, while maintaining all the functionality of the parent without duplication. vmd is written in C++, an object-oriented extension of the C programming language which has been quickly growing in popularity and availability in recent years. Figure 2 here

The design of vmd consists of four main components, each containing a set of objects that work together to accomplish the stated goals of visualization of dynamical molecular data: (1) user interface objects, (2) display objects, (3) molecule objects, and (4) remote simulation objects. The diagram in fig. 2 describes the organization and relationship of the component objects. Boxes within the four components represent software objects. Solid arrows from an originating to a destination object are used to indicate a *uses a* relationship between

objects, meaning the originating object uses the functionality of the destination object in some manner. Dashed lines indicate that the destination objects are inherited from the originating object (*is a* relationship), or contain the originating object (*has a* relationship).

User Interface Objects

Figure 3 shows a snapshot of a vmd session, illustrating the components of the graphics display and graphical user interface. A molecule is shown in the large window in the upper right corner, while other windows represent components of the vmd graphical user interface. A text console for keyboard commands is also provided as a component of the user interface. All vmd commands can be issued via the console as well as through the graphical interface. Command script files can be created and processed by vmd, and the current session can be saved to a log file for future playback. Figure 3 here

Each component of the user interface is a separate object within vmd and performs two basic functions: (1) check for and issue requests for action by the user; (2) report the current state of the system to the user. Each request for action is sent to a central `CommandQueue` processor object, which executes the commands and informs interested user interface components of the action. User interface objects update their state based on this notification of a completed action.

vmd, through the use of a spatial tracking device, provides a three-dimensional user interface in addition to the standard two-dimensional graphical control. A spatial tracking device measures the spatial orientation and position of sensors relative to a fixed source; this allows an application to contain a three-dimensional pointer, which can be operated in

six degrees of freedom (three translational + three rotational). When coupled with a stereo display, the three-dimensional pointer provides the user with a separate three-dimensional user interface which can be employed to request action through the `CommandQueue` object.

Display Objects

For the display of biopolymer structures in `vmd`, two main objects are used: (1) a `Scene`, which maintains a list of all items (`Displayable` objects) that are to be drawn, and (2) a `DisplayDevice`, which can render a given `Scene` to a particular hardware device or, possibly, a file. Each `Displayable` object contains a list of basic drawing commands in a device-independent format. These lists are given to a `DisplayDevice`, which renders them to the selected destination.

The specific code required to draw to a special hardware device, windowing environment, or image file is completely contained within specialized `DisplayDevice` objects, which present a common interface to the rest of the program. Thus, for example, the `Scene` object is not required to know what type of device is being used to draw the image. This design localizes the device-dependent code required to support a particular graphics display method to a single place, which reduces the time and effort required to support a different output device. Rendering to a file is also possible in a method completely transparent to the rest of the program, requiring no changes in other routines. At present, `vmd` contains two `DisplayDevice` options: a standard workstation monitor drawing option using the Silicon Graphics GL library, and a `DisplayDevice` which supports the CAVE [12] virtual display environment. Under development are `DisplayDevice` objects for creating bitmap image files

of the current display, as well as MOLSCRIPT [21] input scripts.

Molecule Objects

A molecule in vmd consists of several elements: a *static structure* describing the atom connectivity, the secondary structure, and the static atomic data such as mass and charge; *dynamic data* such as position, velocity, and energy; and any number of graphical *representations* of the molecule. The source of this data can stem from files of various formats or come directly from an MDComm-brokered connection to a remote simulation. Dynamical data for each discrete time step in a molecular trajectory is stored as an animation list, which can be played back, edited, or saved in a variety of file formats. The file formats currently supported are:

- X-PLOR [8] compatible protein structure (PSF) files (input);
- Brookhaven protein data bank (PDB) files [3] (input and output);
- CHARMm [5] and X-PLOR compatible binary trajectory (DCD) files (input and output).

Each molecule is stored as an instance of a `Molecule` object; these instances are maintained in a `MoleculeList` container object. Each `Molecule` is also a `Displayable`, and is referenced within the `Scene` database.

To visualize the molecular structure, the user can create any number of distinct representations of each molecule, which are updated each animation frame. A representation is a specific rendering method (i.e., CPK, licorice bonds, or simple lines) and specific coloring

method (i.e., by element or residue name) for a selected subset of atoms. vmd contains an atom selection syntax similar to the X-PLOR atom selection syntax [8], which can choose atoms based on name identifiers or position with boolean operators. For example, the command

```
resid TIP3 and within 3 of (resname ALA or resname ARG)
```

selects all atoms in TIP3 residues within 3 Å of an alanin or arginine. Representations for a molecule can be edited, be displayed simultaneously, or be individually turned off.

Remote Simulation Objects

A molecular structure can be retrieved from a remote computer (or the same machine running vmd) through a series of simple user interface controls. Once a connection is established, the user can visualize and interact with the molecule in a manner identical to that used for molecules read from data files. As the trajectory of the molecule is computed by the remote simulation, data for time steps at selected intervals is communicated from the simulation application to vmd via the MDComm software described in section 4. A `Remote` object maintains the state of a connection; each `Molecule` object created via this connection contains a corresponding `Remote` object, and is referenced in a `RemoteList` container object. Any number of MD simulations may be attached to a single vmd process by the user and independently visualized and controlled.

To establish a connection and display a simulation in progress, the following steps are required:

Select Application or Job A host for the simulation is selected, and a list of available MD applications compatible with the MDComm software is retrieved, as well as a list of the currently executing MD jobs on that computer. Either an existing job can be attached to vmd and immediately displayed at the currently executing time step, or a new simulation can be initialized.

Select Parameters To start a new simulation, once a specific application is selected vmd retrieves a list of required and optional parameters for the simulation. Figure 4 shows an example of the simulation parameter editing menu. When all parameters are entered, the simulation program is executed. Figure 4 here

View Results After the connection is made, the static molecular structure is passed to vmd, which creates a new `Molecule` object and begins retrieving dynamic data (i.e., coordinates and energies) from the remote simulation as it becomes available. The new `Molecule` object contains the corresponding `Remote` object for the particular connection. In fig. 3, the molecule shown is a structure from an on-line namd simulation. As the trajectory is calculated, vmd updates the display to show the current state of the simulation. Coordinates may be stored at selected intervals in the animation loop for later playback. vmd also provides the user control over specific simulation parameters such as the current temperature.

2.3 Future Directions

Several improvements and additional features for vmd are under development or planned. For example, new file formats for recording displayed images will be provided, and new

rendering and display options will be added. An interface to the DSSP secondary structure determination program [18] is being developed for accurate display of protein secondary structures. The functionality of the three-dimensional user interface is being extended, in conjunction with development of several new remote simulation control functions.

As discussed, vmd seeks to provide an interactive environment for MD simulations. In progress are enhancements to vmd and the MDComm software to allow a user to add selected forces to specific atoms in a simulated molecule, and to allow a user to make global changes to a molecule such as modification of the structure or translation of atomic coordinates. These interactive changes will be transmitted to the simulation program and incorporated into the simulation directly. Interactive structure building and docking will be made possible by allowing vmd to communicate intended structural modifications and coordinate transformations to a remote simulation. This feature could be used, for example, for interactive placement or refinement of the position of ions or solvent molecules within a protein. Forces applied by the user would move the molecules through a chosen pathway consistent with steric constraints and required rearrangements of surrounding atoms.

3 The Molecular Dynamics Program namd

namd is a parallel molecular dynamics program designed for high-performance simulations in structural biology. The program is highly modular and well documented to facilitate the addition of new algorithms and methods. namd is intended for structural biologists, in particular, those who are attempting large-scale molecular simulations or are interested in testing novel simulation methods. The major objectives of namd are performance, scalability,

and modularity.

Performance A goal for namd is rapid and efficient execution of MD simulations. A second goal is to make possible simulations of molecular systems of a larger size and for a longer time period than has been previously possible. namd was written expressly for message passing, distributed memory parallel machines and can run on several types of massively parallel supercomputers currently available as well as on clusters of high-performance workstations.

Scalability namd is intended for MD simulations of biopolymer systems ranging in size from thousands of atoms to millions of atoms and to be executed on parallel computers with tens to thousands of processors. Thus, namd must scale efficiently with the size of a biopolymer system as well as with the number of processors and must insure scalability in communication and memory usage. This implies conceiving and implementing distributed algorithms for every aspect of the program.

Modularity namd has a modular design to facilitate experimentation with new algorithms or with computational approaches based on new physical concepts. For many calculations of interest, faster hardware, even with massive parallelism, will be insufficient and only new algorithms (including some under development such as [31]) will make certain simulations possible. The achievement of the highest performance, possible through the discovery of new algorithms and methods, requires continuous adaptation of the program. To allow this, namd uses an object-oriented design and employs a high degree of modularity and data abstraction which makes the program easy to modify and maintain.

3.1 Features

namd has several important features which set it apart from other MD programs and also make it easy to use. Since namd is still a very young program, more features are being added, some of which are described in section 3.5. The following features have been implemented in namd.

Full Electrostatics Most MD programs truncate electrostatics to reduce the computational complexity of $O(N^2)$, resulting from summation of pairwise electrostatic forces, to $O(N)$. This type of truncation has been demonstrated to lead to qualitatively wrong descriptions of physical properties, e.g, in the case of membranes [32]. namd has incorporated the Distributed Parallel Multipole Tree Algorithm (DPMTA) [4] which takes the full electrostatic interactions into account. This algorithm also reduces the computational complexity of electrostatic force evaluation from $O(N^2)$ to $O(N)$. As described in section 3.2, DPMTA is combined with a multiple time step scheme to further reduce the computational complexity.

Force Field Compatibility The force field used by namd is the same as that used by the programs CHARMm [5] and X-PLOR [8]. This force field includes local interaction terms consisting of bonded interactions between 2, 3, and 4 atoms and pairwise interactions including electrostatic and van der Waals forces. This commonality allows simulations to migrate between these three programs.

Input and Output Compatibility The input and output file formats used by namd are identical to those used by X-PLOR. Input formats include coordinate files in PDB format

[3], structure files in PSF format, and energy parameter files in the same format used by CHARMm and X-PLOR. Output formats include PDB coordinate files and binary DCD trajectory files. These similarities assure that the molecular dynamics trajectories from namd can be read by CHARMm or X-PLOR and that the user can exploit the many analysis algorithms of the latter packages.

Interoperability with vmd As part of the MDScope system, namd communicates with vmd using the MDComm software. This interaction allows a user to employ vmd as a graphical console which can start namd, view, and modify the simulation to a small extent. The communication between namd and vmd is being enhanced to allow a user to further alter and interact with a running namd simulation.

Dynamics Simulation Options MD simulations may be carried out using several options, including:

- NVE ensemble dynamics;
- NVT ensemble dynamics, using velocity rescaling;
- Langevin dynamics;
- energy minimization.

3.2 Design Overview

The following sections describe the design of namd and how this design accomplishes the goals stated above. The first section outlines the overall algorithms used and the remaining

sections explain how these methods are implemented.

Algorithms

In MD simulations, atomic trajectories are computed from Newton's second law of motion using empirical force fields, such as the CHARMm force field, that approximate the actual atomic force in biopolymer systems. The velocity Verlet integration method [1] is used to advance the positions and velocities of the atoms in time.

To reduce further the cost of the evaluation of long-range electrostatic forces, a multiple time step scheme is combined with the DPMTA method. The local interactions (bonded interactions and electrostatic interactions within a specified distance) are calculated during every time step. The longer range interactions (electrostatic interactions beyond the specified distance) are only computed every k steps, with each set of k steps called a *cycle*. The long-range forces remain the same for each step in the cycle, which amortizes the cost of computing the electrostatic forces over the k steps in the cycle. For appropriate values of k, the error due to holding the forces constant for a few steps is modest compared to the errors incurred from using a finite time step.

The computationally intensive aspect of MD simulations is the evaluation of the force field. One method used by other programs to evaluate the force field in parallel involves dividing the interactions among processors using various forms of force decomposition [6, 26]. The problem with these force decomposition schemes is their scalability. While the computational complexity for each processor in force decomposition can be reduced to $O(N/P)$, the communication complexity is $O(N)$ in the naive case and $O(N/\sqrt{P})$ [26] in the best

case. This limits the scalability of such methods to large numbers of processors.

To avoid this limitation, namd uses spatial decomposition of the biopolymer system. Spatial decomposition reduces both the computation and communication complexity of calculations performed by each processor to $O(N/P)$ [11]. This results in better scalability than force decomposition schemes that require summation across all processors. Figure 5 here

A long-standing problem with spatial decomposition is load balancing. A spatial decomposition that evenly distributes the computational load causes the region of space mapped to each processor to become very irregular, hard to compute and difficult to generalize to the evaluation of many different types of forces. namd addresses this problem by using a simple uniform spatial decomposition where the entire model is split into uniform cubes of space called *patches*. Each patch is slightly larger than the local interaction distance. This requires that there be many more patches than processors (as shown in fig. 5), but results in each patch needing to share information only with its immediate neighbors. A sample of the spatial decomposition for a small polypeptide is shown in fig. 6. Since each patch is a simple, uniform cube, it is easy to compute the decomposition and to use it as a framework for many types of calculations. Figure 6 here

Message-Driven Execution

Patches are responsible for calculating the forces acting on their local atoms. These interactions require coordinate information for not only the local atoms, but for atoms from neighboring patches as well. If each processor were assigned a single large patch, the processor would repeatedly have to stop and wait for information to arrive from neighboring

patches. During these periods, the processor would be idle. These wait periods can be caused not only by network communication latency, but also by work load differences between patches. The sending patch may be busy with some other task and, thus, a significant delay may arise before the necessary data is sent.

In order to reduce the amount of idle processor time, namd uses a multi-threaded execution model. Each patch acts as its own thread of control, and each processor is assigned multiple patches. These patches are then scheduled in a message-driven manner that reduces the amount of idle time on the processor. Message-driven implies that a thread is only scheduled to run when a message has arrived for that thread. A graphical depiction of how this scheduling helps potentially reduce idle times is shown in fig. 7. The figure shows the utilization of a processor for single-threaded and multi-threaded execution of patches. In the single-threaded case, after performing some calculations the processor waits for messages that are needed to calculate more interactions, during which time the processor is idle. In the multi-threaded case, however, while waiting for messages for a particular patch, the processor can switch to another patch for which a message has arrived. This overlaps the wait time for one thread with the computation time of another thread, thus reducing the idle time of the processor. Figure 7 here

To accomplish the multi-threaded execution, each patch must be designed such that it can process each incoming message independently, and in an arbitrary order. Each processor continuously executes a message-driven loop which receives all messages, determines the patch that the message belongs to, and then yields control to the patch. The patch resumes execution, consumes the message, and when finished with the message returns control.

Object-Oriented Design

Like the program vmd, namd uses an object-oriented design and is implemented in C++. The program consists of objects (or *classes*) that have their own data and functions to operate on this data. Other objects can not directly access the local data of an object, but instead must use the defined functions (*methods*) of the object. The only means for interacting with a well designed object is through its methods. Thus, the internal implementation of these objects is hidden from the rest of the program. While this type of design can be simulated in traditional programming languages, by using a truly object-oriented programming language such as C++ these ideas are naturally implemented and enforced.

Portability problems and performance penalties that may be incurred by the use of C++ compared to C or FORTRAN is an important issue currently under discussion [16]. Because of the possible performance penalties, namd avoids excessive use of features that may degrade performance, e.g., run-time function table lookups as arise through the use of virtual functions. A key problem limiting performance in C++ is the extensive use of temporary variables resulting from complex expressions, which reduces the number of compiler optimizations that may be applied and which lead to inefficient use of registers and cache memory. namd pays particular attention to this issue by using standard C constructs for the central computation loops, and rational design of C++ classes which avoid temporary variable allocation/deallocation. Also, while C++ is becoming available on most machines, an effort was made to avoid some of its features, such as templates, that may not be implemented on all machines. With these precautions, the advantages gained in modularity using C++ significantly outweigh porting or performance problems that may arise.

Class Structure

Figure 8 here

Following the ideas outlined in the previous section, namd is composed of a simple set of classes that are used to provide the necessary components for the simulation. Figure 8 shows the objects that are present on each processor running namd, and table 1 gives a description of the purpose of each of the objects in this diagram. There are three global objects on each processor: (1) the `Communicate` object, a protocol-independent agent for inter-processor communication; (2) the `Inform` object, a transparent means of printing messages to the screen from any processor; (3) the `Node` object, which contains all of the important objects for the simulation. The `Node` object includes objects that provide static data such as the `SimParameters`, `Molecule`, and `Parameters` objects as well as objects that provide global state data or services such as the `Collect`, `Output`, `PatchDistrib`, `LoadBalance`, and `FMAInterface` objects. However, the most important member of the `Node` object is the `PatchList` object. The `PatchList` object contains all patches owned by the processor, as well as the multi-threaded logic used to schedule the operations of these threads. It is within the patches contained in this object that the calculations needed for the simulation are performed.

Table 1 here

The `Patch` objects perform the actual simulation. The structure of a `Patch` object is shown in fig. 9. Each `Patch` object is responsible for one of the cubic regions of space determined by the spatial decomposition of the biopolymer system. Each patch is responsible for calculating the forces and performing integration on all the atoms in its region of space. Thus, each `Patch` object contains the data for each atom including position, velocity, and forces, a set of force objects that are each responsible for the computation of some component

of the force field, and logic that controls the execution of the force objects upon receipt of a message from the `PatchList` object. Each of the force objects has a similar interface to the `Patch` object. This allows the addition or modification of components to the force field without altering anything but the control logic of the `Patch` object. Figure 9 here

Parallel Programming Model

An important characteristic of namd is the separation of the parallel control logic modules from the sequential computational modules. The parallel control logic modules deal with the sending and receiving of messages and with the invocation of the appropriate sequential module to process the message. The nature of a message-driven design facilitates this separation and provides a clean interface between parallel control and sequential computations. In the message-driven design, messages are received by the parallel logic modules only. These modules invoke the sequential modules. A new sequential module can be integrated by simply providing interface functions to the parallel logic modules. Thus, the sequential modules are separated from any details of how messages are received and from the scheduling of execution of the sequential modules. This is very natural in C++, where the sequential modules are objects which have methods that act on specific information.

The separation of message receipt and scheduling from the sequential computation makes namd portable to different parallel systems, since the details of the communication protocols used are isolated to a few modules. Currently, the parallel control modules for namd have been implemented using PVM [14] and Charm++ [20].

The PVM implementation is built upon a generic implementation applicable for any

send and receive message passing library such as PVM, TCGMSG, CMMD, etc. It is implemented using a `Communicate` class which provides protocol-independent send and receive mechanisms. The parallel control modules are then built to send and receive messages using this class and to schedule the processing of these messages in a message-driven manner. The `Communicate` class is currently implemented using PVM, but can be ported to any such receive-based method by simply changing the protocol-specific send and receive calls.

The other implementation of the parallel control logic is based on Charm++ [20], a portable object-oriented message-driven parallel programming environment developed by one of the authors. The parallel control logic is expressed more naturally in Charm++ than it is with a receive-based language. A significant advantage of programming in a message-driven language is modularity and compositionality. Multiple parallel modules that are developed independently can be combined without performance penalties. The flow of control can switch back and forth between modules automatically, thereby, overlapping useful computations in one module with potential idle times in the others. A receive-based message passing system, on the other hand, cannot achieve this as elegantly or efficiently as a message-driven system [19].

Full Electrostatics Interface

As previously stated, namd incorporates a module that uses a parallel multipole algorithm to evaluate the full electrostatic interactions. The interface between namd and DPMTA is a prototype for other external modules that will be incorporated into namd. This interface is a simple set of sequential procedure calls. However, since namd and DPMTA do not share

common data decomposition schemes, intervening scatter/gather operations are required. This is accomplished through *representative* objects that interface to namd and DPMTA. The representative objects that interface with namd are located in each `Patch` object and are responsible for gathering the coordinate information from each patch, and for passing that information to the DPMTA representative. The DPMTA representative invokes DPMTA to calculate the electrostatic results and then distributes the results back to the namd representatives in each patch for use in the simulation. This interface uses the representatives to compensate for the differences between the implementations of namd and DPMTA and forms a model that can be used for the incorporation of other external modules that differ in their decomposition from namd.

3.3 Documentation

To achieve the objective that researchers can quickly and easily understand and alter namd, a programmers guide has been maintained throughout the development of the program. The documentation provides a description of the design and implementation of the program, and explains not only what is implemented and how, but also why it was done in a particular way.

The programmers guide includes sections on the force field, overall design, source code conventions, and implementation. The first section contains mathematical details of exactly what the program calculates, including the energy and force equations for each component of the force field. The design section describes the design of the program from the choice of C++ through the details of the spatial decomposition scheme used. It includes details of the

design as well as the reasoning behind it. The source code conventions section establishes guidelines for adding code to namd. This insures that the source code maintains a consistent look and feel even though many programmers may be involved. The implementation section is the largest in the guide. It details the exact interface to and operation of each of the objects or modules in namd. With the modularity provided by C++ objects and the definition of each interface, adding or changing namd can be done rapidly and in a consistent manner.

3.4 Current Status

While still early in its development, namd is a functional molecular dynamics program. A force field compatible with that used by CHARMm and X-PLOR is complete. The program has been tested by comparing with results from X-PLOR for a small 12-residue polypeptide (66 atoms), the bovine pancreatic trypsin inhibitor (898 atoms), the protein bacteriorhodopsin (3,762 atoms), as well as a complex of water, a 17 base-pair oligonucleotide, and a dimer of the glucocorticoid receptor DNA binding domain (13,566 atoms altogether).

Without any performance tuning at this point, namd running on one HP-735/125 processor is about 1.75 times slower than X-PLOR on the same machine for the simulation of bacteriorhodopsin. The graph in fig. 10 indicates the parallel performance of namd, by plotting the speedup as compared to a two-processor simulation for the calculation of the glucocorticoid receptor DNA binding domain system with and without DPMTA. Since the speedup shown in fig. 10 is relative to the speed of a simulation using two processors, the graph indicates an approximately 50 % speedup for up to eight processors. A number of performance bottlenecks such as load balancing have been identified, and it is expected that

performance tuning of namd will significantly improve these performance values. Figure 10 here

3.5 Future Directions

There is still a large amount of development to be completed, foremost, performance testing and tuning. A major part of performance tuning involves the load balancing strategies discussed below. Another important item will be the porting of namd to many platforms. Versions of namd exist for clusters of HP workstations, the Convex Exemplar, and the Intel Paragon. Versions for the SGI Power Challenge, Cray T3D, IBM SP2 and other machines are planned. An effort will be made in converting any algorithms that are currently centralized to a distributed algorithm, including I/O. This will be critical in scaling efficiently on machines with hundreds or thousands of processors.

Load balancing is a key issue for achieving high performance with namd. The goal is to find a distribution of patches such that total execution time is minimized. This can be achieved by keeping all processors busy with useful work while minimizing the communication across processors. Load balancing is done in two stages. The first is the initial distribution of patches to processors at the beginning of the simulation. The second stage is the incremental update of this distribution to keep the system balanced as atoms move between patches. Different algorithms can be applied during these two stages. Particularly, an incremental load balancing algorithm can be more efficient by focusing on the small changes in an already well balanced system. Currently, an interface to a generic load balancing module is defined and implemented and various load balance decision algorithms are under development.

As has been emphasized, a goal of namd is to provide a testbed for new methods and

algorithms for molecular dynamics. Accordingly, prominent in the future development will be the addition of new simulation features, for example, the simulation of NpT ensembles [23]. More elaborate multiple time stepping algorithms will be experimented with, and periodic boundary conditions and free energy perturbation calculation capabilities will be included. The interface with vmd will be enhanced to allow interactive simulations for structure prediction.

4 Linking vmd and namd with MDComm

Molecular dynamics applications are particularly well-suited to a visual computing implementation, with an extensive suite of techniques for graphical representation of molecular structures in common use. Furthermore, the network bandwidth required to efficiently couple an MD simulation with a graphical front-end is not prohibitive; spatial coordinates of molecular systems of up to 87,000 atoms require less than 1 MB of single-precision data. A simple estimate of the sustained I/O rate r (in bytes/second) which can support the data requirements of an MD simulation of N atoms, which needs t seconds to compute each time step, is $r = 12N/t$.

Recent studies of the performance of various protocol combinations over local area ATM and Ethernet networks indicate that Berkeley stream sockets can sustain slightly over 2 MB/s over ATM and slightly over 1 MB/s over Ethernet networks [22]. At these transfer rates, the computational rates of MD applications on current hardware platforms do not tax the capacity of available networks. In addition, through control of the frequency of interactions between the application and visualization programs, it is possible to better match varying

network conditions.

MDCOMM is the enabling software which connects namd and vmd in a single computational environment. It provides a mechanism for using vmd to visualize the results of other MD applications or, conversely, to drive another visualization front-end with namd. MDCOMM is as hardware- and software-independent as possible and is operational using current technology. This section describes the features, implementation, and status of MDCOMM.

4.1 Features

MDCOMM focuses on those functional components required to allow cooperative execution of a scalable MD program and an interactive visual interface. It is a compact software layer that provides the following features:

- process control in a networked environment;
- multiple simultaneous connections to simulations;
- asynchronous connections to in-progress simulations;
- support for heterogeneous systems;
- concurrency on multiprocessor systems;
- low-overhead implementation;
- application-independent design;
- comprised of freely-available and stable software components that are available on (or portable to) a wide variety of UNIX platforms.

Figure 11 shows the architecture of the system, depicting a connected instance of namd and vmd. Figure 11 here

4.2 Implementation

The relationship between a simulation program and a visualization front-end is primarily that of a *producer-consumer* pair. In the context of namd and vmd, namd produces data such as atomic coordinates in three-dimensional space, energies, temperature, etc. These quantities are consumed by vmd and used to generate an appropriate visual representation. Data that describes the status of namd itself is also of interest, e.g., the cumulative CPU time or the current decomposition (patch assignments). Most of the data that flows from namd to vmd is *dynamic*; the quantity associated with a particular object varies with each time step, e.g., an atom's location changes over time or its assignment to a patch changes as the simulation proceeds. Data used to "configure" vmd does not change over time; these latter quantities are *static*; examples include the connectivity of a molecular structure or characteristics of component objects such as atom or residue names.

A mechanism for transfer of static and dynamic data resides at the lowest level of the MDComm software and is used internally, to implement MDComm coordination protocols, and externally, to be accessible to namd and vmd. To support producer/consumer pairs which may execute on machines with differing number representations, these routines must support automatic conversion of data. MDComm software incorporates the XDR (eXternal Data Representation) standard [28] for data conversion between heterogeneous systems. For the actual application layer, MDComm employs the BSD stream socket implementation of

the TCP (Transmission Control Protocol) standard. TCP and XDR implementations are available today on most systems used by the structural biology community. As well, vendors of current-generation distributed-memory multiprocessors often provide node-level support for socket-style communication primitives.

Software Organization

An earlier implementation of visual molecular dynamics followed a strict consumer-producer model, incorporating all communication directly into the MD program and graphical front-end. From an organizational perspective, these two programs operated as peers that were responsible for all communication and control protocols and employed *send* and *receive* primitives available in the MDCOMM library. To provide increased flexibility and functionality, a division of labor between multiple software components was necessary. The current implementation adopts a very different philosophy: MDCOMM software now occupies a *supervisory* role with respect to namd and provides one or more cooperative consumer coprocesses available to vmd.

A general-purpose program, the *manager*, provides several services pertinent to its *home computing system* (the computer on which the manager executes): it advises potential clients of the availability of applications, monitors the set of active daemon-application pairs, and maintains a current list of protocol port assignments for active daemon-application pairs. Shown as chemd in fig. 11, the manager is independent of the application domain; it can provide services for any type of application that could be sensibly placed under the control of MDCOMM. chemd is implemented as a Remote Procedure Call (RPC) server using the

rpcgen protocol compiler.

The *daemon*, shown as namdd, acts as a parent and intermediary between a single instance of an application and potential clients. Status information is also passed from the daemon to the manager, which then is able to provide the necessary data to potential clients that wish to interact with an MDComm daemon and its application. The daemon caches all static data associated with an application in order to provide "setup" data to a client. Finally, the daemon receives all interactive commands from a connected client and relays the command and arguments to its associated application.

The *application* is a child process of the daemon and is the producer in the producer-consumer relationship described earlier. namd is an example application incorporated under MDComm. The application is responsible for providing all static data to the daemon after initialization; thereafter, it provides dynamic data (as it becomes available) to a consumer. Periodically, the application checks for client requests which have been relayed through the daemon; MDComm provides several built-in requests as well as a mechanism for tailoring application-specific responses to requests (in essence, the application can define "callbacks" which are invoked when certain client-generated events occur).

The *client*, vmd in fig. 11, may initiate daemon/application pairs and utilize data generated by one or more such pairs. The client also has access to several remote procedure calls that are serviced by the manager; it may inquire about the applications that are available on a remote system or request the necessary information to establish a connection to an existing daemon/application pair. Through MDComm-provided or user-defined routines, the client also can interact with the application, for example, altering simulation parameters, changing

the interval between transfers of dynamic data, terminating a connection or directing the application to terminate. From the client's perspective, a *connected application* is a triple consisting of the daemon, the application, and the consumer. A client may have many such triples available at any time. Only one connected application is shown in fig. 11.

The *consumer*, md_consumer, plays a limited role: it accepts data as it becomes available from the application. The consumer places the data in an area of shared memory so that it may be used by the client; if data is XDR-encoded, the consumer deserializes the data as well. Consumers are created by the client for each instance of a connected application and exist until a client disconnects from an application or until an application terminates. On a multiprocessor system, the consumer's activities can proceed concurrently with the client's, overlapping I/O and data conversion with user interaction and graphics processing.

Example

The relationship between vmd, namd, and MDComm can be further described through an example. The user starts at the console of a graphics workstation on which vmd is available. After startup, vmd's remote execution form allows selection of another network-accessible host. Through MDComm, vmd can query chemd for the names of available applications on the remote host. chemd has a database of installed MDComm-compatible applications from which names are returned to vmd. After selecting one of the applications, the user still has to determine how to correctly execute the application. In other words, after asking "*What* can I run on this host?", the next question is "*How* do I run it?". Again, chemd supplies this information to vmd, by returning a list of user-selectable options which are contained in

the database entry corresponding to the selected application.

At this point, vmd still operates as a single process, although it has now obtained all information necessary to initiate an MD simulation on the remote system. Using vmd's graphical interface, the user specifies all the parameters of the simulation and submits a request for remote execution (as shown in fig. 4).

Once the user submits the job, MDComm provides two modes for job initiation: *immediate* or *deferred* connection. In either case, the following events take place:

1. An instance of the daemon, namdd, is initiated using *exec* or *rexec*.

2. The user-defined argument list is passed to namdd, which marshals the arguments and initiates an instance of namd. namdd registers with chemd, providing information about the application and system-assigned protocol ports that can be used to request connections by potential clients. namdd then receives all static data from namd; subsequently, this data can be relayed to prospective clients.

Upon request by the user, an immediate connection is created through:

- relay of static data to vmd from namdd;
- creation of a shared memory segment on the client system to receive dynamic data;
- initiation of md_consumer, which will manage the shared memory segment;
- establishment of an active socket between the new md_consumer and namd.

All components of an active connection have now been created and can share the data

needed by vmd. As data arrives from the application, mutual exclusion is guaranteed by SYSV IPC semaphores that are created when md_consumer begins execution.

In the case of a deferred connection, namdd and namd begin execution without any communication to external clients. A connection can be established at a later time; all information is kept by chemd and available through MDComm remote procedure calls. Prospective clients can obtain a list of currently executing namdd/namd pairs and the assigned protocol port where namdd is listening for connections. If no active connection exists between namd and a client, the second procedure above is followed to establish the requested connection. Table 2 summarizes the facilities provided by MDComm. Table 2 here

4.3 Status

MDComm is implemented using the C programming language (with support for ANSI- and non-ANSI compliant compilers) and is operational today on a variety of systems, including Silicon Graphics, Hewlett-Packard, Convex Exemplar, and Sun computers. namd is the third application that has been incorporated into a networked environment using MDComm software.

At the present time, only one-to-one communication is supported between an application and consumer, introducing a severe scalability problem. Modifications to support a many-to-one model, where multiple application node programs can provide data to a consumer, will alleviate this bottleneck.

As mentioned in Sections 2.3 and 3.5, enhancements to the control capabilities of the client are also being designed to allow for additional interactive control of applications.

MDComm has evolved significantly during its development to date. To provide mature, production-quality software, close attention will be given to fault-tolerant features which are necessary in a networked environment. At present, error handling and recovery are provided only to a small extent; these aspects will be improved in the next phase of development.

5 Structure Refinement and Simulation of the Poliovirus Coat

As stated above, MDScope allows simulations of very large biopolymer systems as well as interactive structure building and refinement. A uniquely suitable system which can take advantage of the capabilities of MDScope is the coat of the poliovirus, the focus of an ongoing group research project. This protein complex, shown in its entirety in fig. 12a, forms an icosahedral shell of 300Å diameter and envelopes the viral RNA. The coat consists of 240 icosahedrally arranged proteins with 498,000 atoms altogether. Figure 12b presents a pentamer unit of the icosahedral coat and fig. 12c illustrates the four proteins, VP1 through VP4, which make up one of the five segments in the pentamer. Obviously, simulation of this complex poses a challenge.

Figure 12

The poliovirus structure has been determined by X-ray diffraction [17]. The structure here reveals a site which plays a key role in the infection process as various antiviral drugs can replace a sphingosine molecule which naturally occupies the site. It is believed that a lack of sphingosine alters the stability of the virus coat, leading to a swelling of the coat and the creation of windows through which the viral RNA can exit into the then infected cell [15] .

The sphingosine is actually contained in VP1, as shown in fig. 12d. Antiviral drugs replacing sphingosine do not leave the site and by their presence prevent the swelling and fenestration of the coat. At present it is not understood how sphingosine controls the coat's swelling and fenestration and it is hoped that MD simulations can help elucidate the mechanism.

5.1 Interactive Molecular Modelling

The structure of the virus coat was not completely resolved by X-ray diffraction; a few segments of the coat proteins were missing. To complete the structure, the amino acid sequences of the missing protein segments were compared with a representative set of proteins in the protein data bank and homologous sequences found [2]. The respective structures were then used to complete the coat proteins.

Structures determined from homology matches are not often correct, as amino acids can take on any one of several conformations. The large-scale differences which must be effected to allow the backbone of the new structure to fit into its proper location, can often be corrected by changing a small number of dihedral angles. Sterical conflicts can then be resolved through minimization and equilibration using MD simulations.

For the poliovirus coat described here, refinement has been previously carried out using Quanta [27]. The motions of the two main input devices, the mouse and an on-screen dial box, did not readily map to the actions needed to adjust a three-dimensional structure. One of the goals of MDScope is to enable structure building in a three-dimensional environment. For this purpose vmd provides the user with two three-dimensional pointers to separately move two structural elements relative to each other. Coupled to a stereo display device, this

provides an intuitive means by which to examine the quality of an initial guess of a structure.

Most structural changes occur in the first one hundred steps of minimization. An effective modelling program must be able to complete this minimization sufficiently fast, i.e., in about ten seconds, to allow a natural user response. This time scale presently limits the minimization and equlibration to systems on the order of a thousand atoms, an acceptable size as most structure building can be limited to small volumes. Once a relevant region of space is defined, namd will treat atoms outside that region as fixed while minimizing those inside.

5.2 Simulation

Molecular dynamics simulations of virus coat proteins will investigate how sphingosine affects the conformation of its immediate protein environment. This requires a simulation of at least 20,000 atoms for about a nanosecond. It is not possible to use the symmetry of the crystal structure to reduce the problem size as the onset of infection occurs when a cellular receptor binds to one of the coat proteins, placing the virus into an asymmetrical environment.

To complete a simulation of the necessary scale in a reasonable time, computations must be performed in parallel, a task for which namd is specifically designed. Its spatial decomposition avoids communication bottlenecks seen in other algorithms, promising a nearly linear speed-up with processor number, and its scalable data distribution provides effective use of computer memory.

6 Availability

vmd, namd, and MDCOMM may be obtained free of charge for educational or non-profit use, including all source code and documentation. Complete information about namd and vmd is accessible from the Theoretical Biophysics World Wide Web (WWW) server, available via URL `http://www.ks.uiuc.edu`.

vmd is available for Silicon Graphics workstations running IRIX version 4.0.5 or later, or Hewlett-Packard PA-RISC workstations runninng HPUX version 9.01 or later. vmd requires a C++ compiler and the use of the GL graphics library; on Hewlett-Packard workstations vmd supports the use of the NPGL graphics library, a port of the GL library to many workstation architectures available from Portable Graphics, Inc. The full source code for vmd with all features described here can be obtained via anonymous ftp from the Theoretical Biophysics ftp server `ftp.ks.uiuc.edu`, in the directory `pub/mdscope/vmd`. This distribution also includes a users guide, a programmers guide, and installation guide for researchers interested in both using the current program, or adding or changing features.

The complete namd source code may be obtained via anonymous ftp from the Theoretical Biophysics ftp server `ftp.ks.uiuc.edu`, in the directory `pub/mdscope/namd`. This distribution also includes the programmers guide. namd requires a C++ compiler and an underlying communication package. Currently PVM and Charm++ are the supported communications packages. PVM may be obtained via anonymous FTP from `ftp.netlib.org`, in the directory `pvm3`. Charm++ may be obtained from `a.cs.uiuc.edu`, in the directory `pub/CHARM++`. The DPMTA code used for full electrostatics may be obtained from `ee.duke.edu`, in the directory `pub`.

The MDComm software, including the namd daemon, manager, consumer, and library, is available for Silicon Graphics, Hewlett-Packard, Convex Exemplar, and Sun computers. MDComm is implemented in C, with support for ANSI- and non-ANSI compliant compilers). Source code for MDComm may be obtained via anonymous ftp from the Theoretical Biophysics ftp server `ftp.ks.uiuc.edu`, in the directory `pub/mdscope/mdcomm`.

7 Acknowledgements

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 the MDScope software. 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 ASC-8902829), and the Roy J. Carver Charitable Trust.

Intel Paragon and Connection Machine 5 access has been made possible through the help of the San Diego Supercomputer Center (SDSC) and the National Center for Supercomputing Applications (NCSA), and a MAC computer usage grant (MCA93S028P). The authors wish to thank NCSA for access to the Convex Exemplar parallel computer.

References

[1] M. P. Allen and D. J. Tildesley. Computer simulation of liquids. Oxford University Press, New York, 1987.

[2] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. *J. Mol. Biol.*, 215:403–410, 1990.

[3] F. C. Bernstein, T. F. Koetzle, G. J. B. Williams, J. E. F. Meyer, M. D. Brice, J. R. Rodgers, O. Kennard, T. Shimanouchi, and M. Tasumi. *J. Mol. Biol.*, 112:535–542, 1977.

[4] J. Board, Z. Hakura, W. Elliot, and W. Rankin. Scalable variants of multipole-accelerated algorithms for molecular dynamics applications. Technical Report TR94-006, Duke University, Dept. of Elec. Engr., 1994.

[5] B. R. Brooks, R. E. Bruccoleri, B. D. Olafson, D. J. States, S. Swaminathan, and M. Karplus. *J. Comp. Chem.*, 4(2):187–217, 1983.

[6] B. R. Brooks and M. Hodoscek. *Chemical Design Automation News*, 7(12):16–22, 1992.

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

[8] A. T. Brünger. 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, 1992.

[9] M. Carson. *J. Appl. Cryst.*, 24:958–961, 1991.

[10] M. Carson and J. Hermans. In J. Hermans, editor, *Molecular Dynamics and Protein Structure*, pages 165–166. University of North Carolina, Chapel Hill, 1985.

[11] T. Clark, R. Hanxleden, J. McCammon, and L. Scott. In *Proceedings of the Scalable High Performance Computing Conference*, Knoxville, TN, May 1994.

[12] C. Cruz-Neira, D. J. Sandin, and T. A. DeFanti. In *Proceedings of SIGGRAPH '93*, page 135, Anaheim, CA, 1993. Association for Computing Machinery.

[13] T. E. Ferrin, G. S. Couch, C. C. Huang, E. F. Pettersen, and R. Langridge. *J. Mol. Graphics*, 9:27–32, 1991.

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

[15] R. A. Grant, C. N. Hiremath, D. J. Filman, R. Syed, K. Andries, and J. M. Hogle. *Curr. Biol.*, 4(6):784–797, 1994.

[16] S. W. Haney. *Computers in Physics*, 8(6):690–694, 1994.

[17] J. M. Hogle, M. Chow, and D. J. Filman. *Science*, 229:1358, 1985.

[18] W. Kabsch and C. Sander. *Biopolymers*, 22:2577–2637, 1983.

[19] L. V. Kale and A. Gursoy. In *Proceedings of the* 7^{th} *SIAM Parallel Processing Conference*, San Fransisco, CA, Feb. 1995.

[20] L. V. Kale and S. Krishnan. In *Proceedings of OOPSLA-93*, Washington, DC, Sept. 1993.

[21] P. Kraulis. *J. Appl. Cryst.*, 24:946–950, 1991.

[22] M. Lin, J. Hsieh, D. H. C. Du, J. P. Thomas, and J. A. MacDonald. In *Proceedings of Supercomputing '94*, pages 154–163, Washington, DC, 1994. IEEE Computer Society Press.

[23] G. Martyna, D. Tobias, and M. Klein. *J. Chem. Phys.*, 101:4177–4189, 1994.

[24] J. A. McCammon and S. C. Harvey. Dynamics of proteins and nucleic acids. Cambridge University Press, Cambridge, 1987.

[25] Minnesota Supercomputer Center, Inc., Minneapolis, MN. Xmol, version 1.3.1, 1993.

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

[27] Polygen Corporation, 200 Fifth Av., Waltham, MA 02254. Quanta, 1988.

[28] Sun Microsystems. XDR: External data representation standard. RFC-1014, 1987.

[29] W. F. van Gunsteren and H. J. C. Berendsen. Gromos manual. BIOMOS b. v., Lab. of Phys. Chem., Univ. of Groningen, 1987.

[30] P. K. Weiner and P. A. Kollman. *J. Comp. Chem.*, 2:287, 1981.

[31] G. Zhang and T. Schlick. *J. Comp. Chem.*, 14:1212–1233, 1993.

[32] F. Zhou and K. Schulten. *J. Phys. Chem.* In press.

Class Name	Description
`Communicate`	Protocol independent means of message passing, including operations such as send, receive, and broadcast
`Inform`	Print messages to the screen from any processor
`SimParameters`	Container class for static simulation data such as number of time steps, time step size, etc.
`Molecule`	Container class for static structural data for the molecule such as which atoms are bonded by various types of bonds, explicit exclusions, etc.
`Parameters`	Container class for energy parameters from the parameter files
`LoadBalance`	Determine the distribution of patches to processors that will keep the load balanced across all processors
`Collect`	Gather global information such as energy totals
`Output`	Produce all forms of output for namd such as trajectory files, energy output, etc.
`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`	Contains all the patches belonging to this processor and the logic to schedule the execution of these patches

Table 1: Description of the purpose of the objects present on each processor in namd.

	Manager	Daemon	Application	Client	Consumer
Application/Process Query	X			X	
Process Control/Execution		X		X	
Application Definition	X			X	
Data Access				X	X
Static Data Transfer		X	X	X	
Dynamic Data Transfer			X		X
Data Conversion			X		X
Application Steering/Event Processing		X	X	X	

Table 2: Summary of the facilities provided by MDCOMM and the components of an MDCOMM application which are involved in each facility.

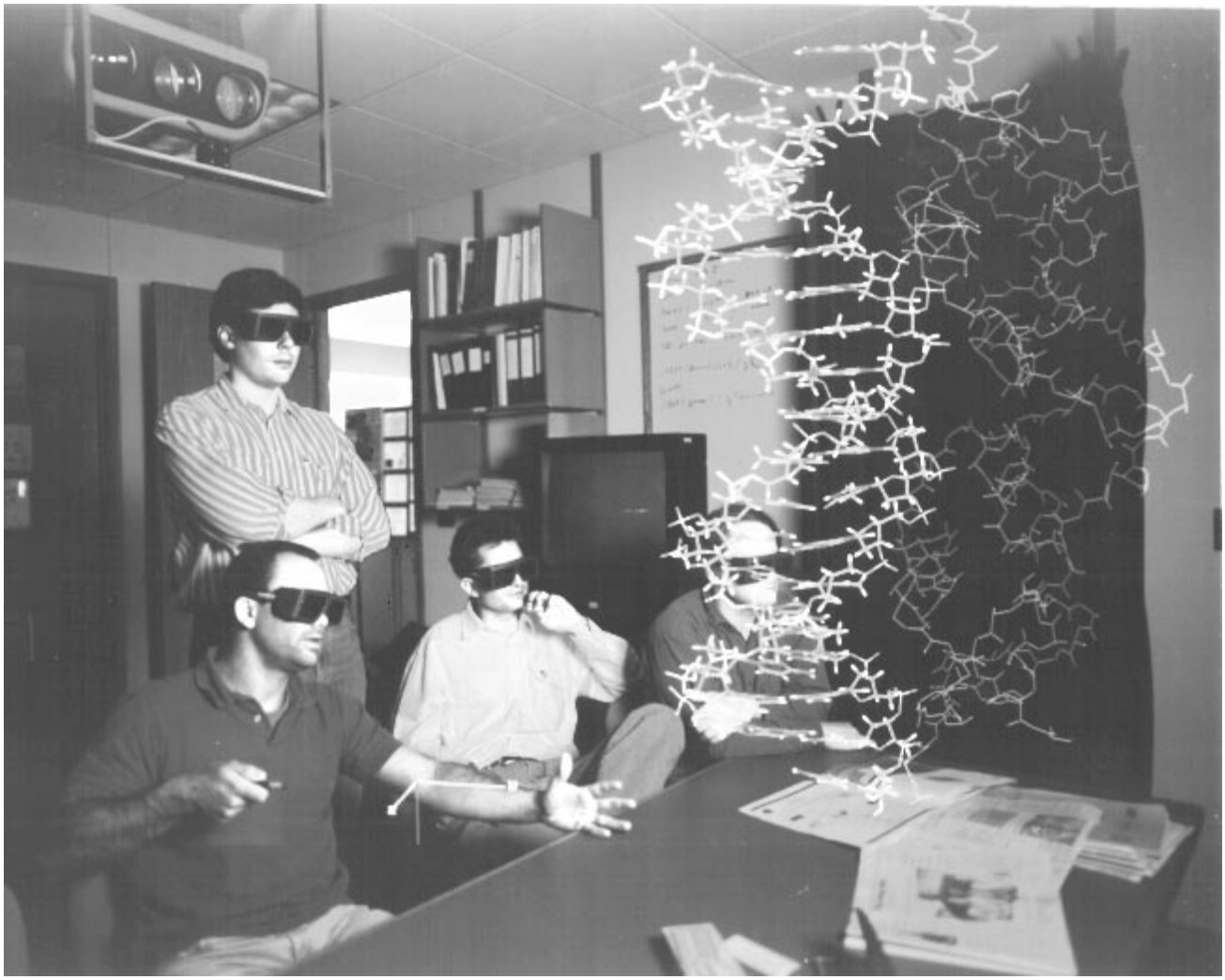

Figure 1:
Researchers of the Theoretical Biophysics group at the University of Illinois and the Beckman Institute utilizing a stereo projection facility with MDScope software, discussing the structure of a protein-DNA complex. (Photo courtesy of Rich Saal of the Illinois State Journal-Register, Springfield, Illinois.)

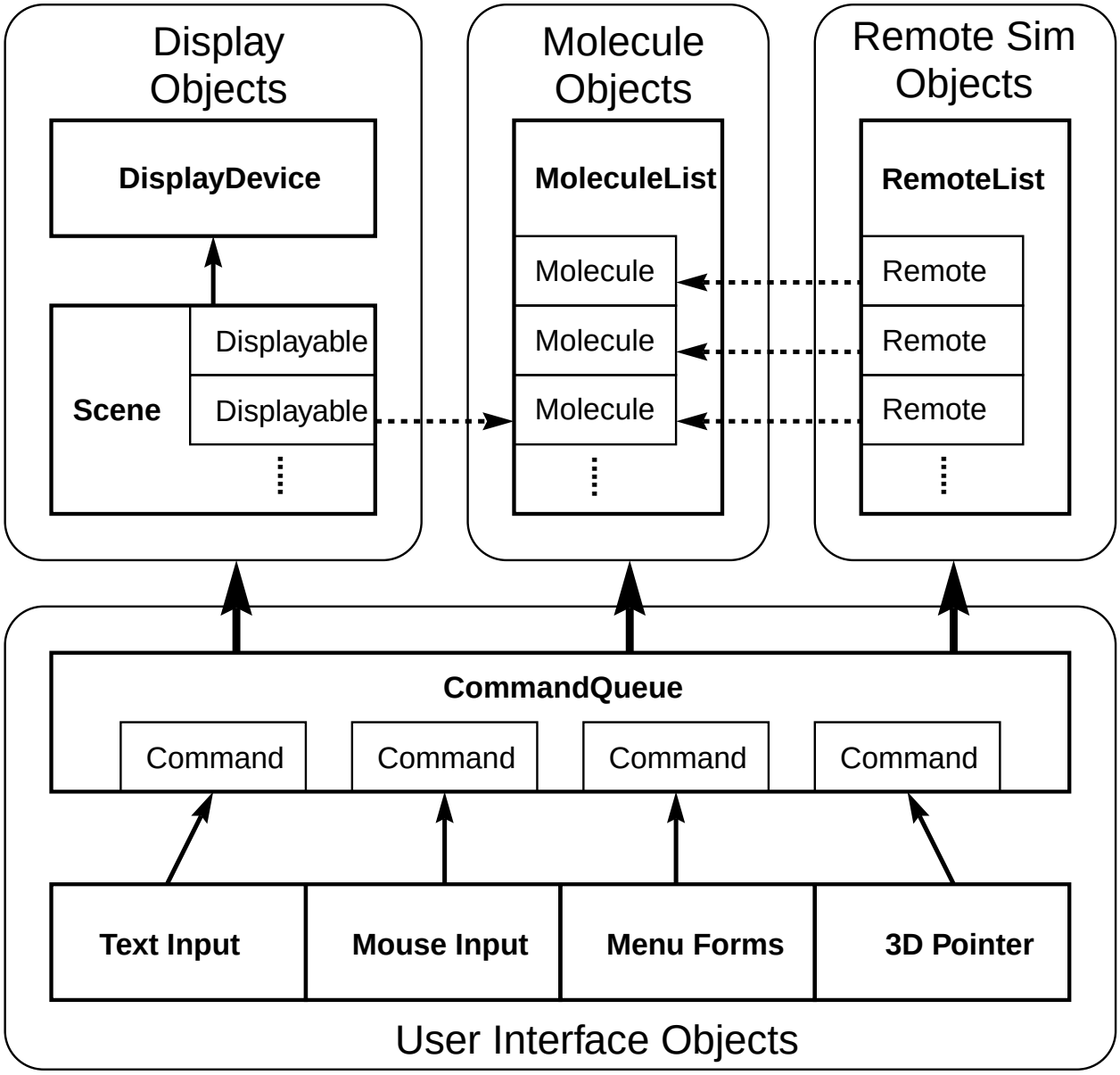

Figure 2:
Component object diagram for vmd. The program is comprised of four main categories of objects: Display objects, responsible for image rendering; molecule objects, which maintain the individual molecular structures and dynamic data; remote simulation objects, which maintain the state connections with remote computers; and user interface objects, which accept and act upon commands from the user. Solid arrows between objects and categories indicates a *uses a* relationship; dashed arrows between objects indicates an *is a* relationship.

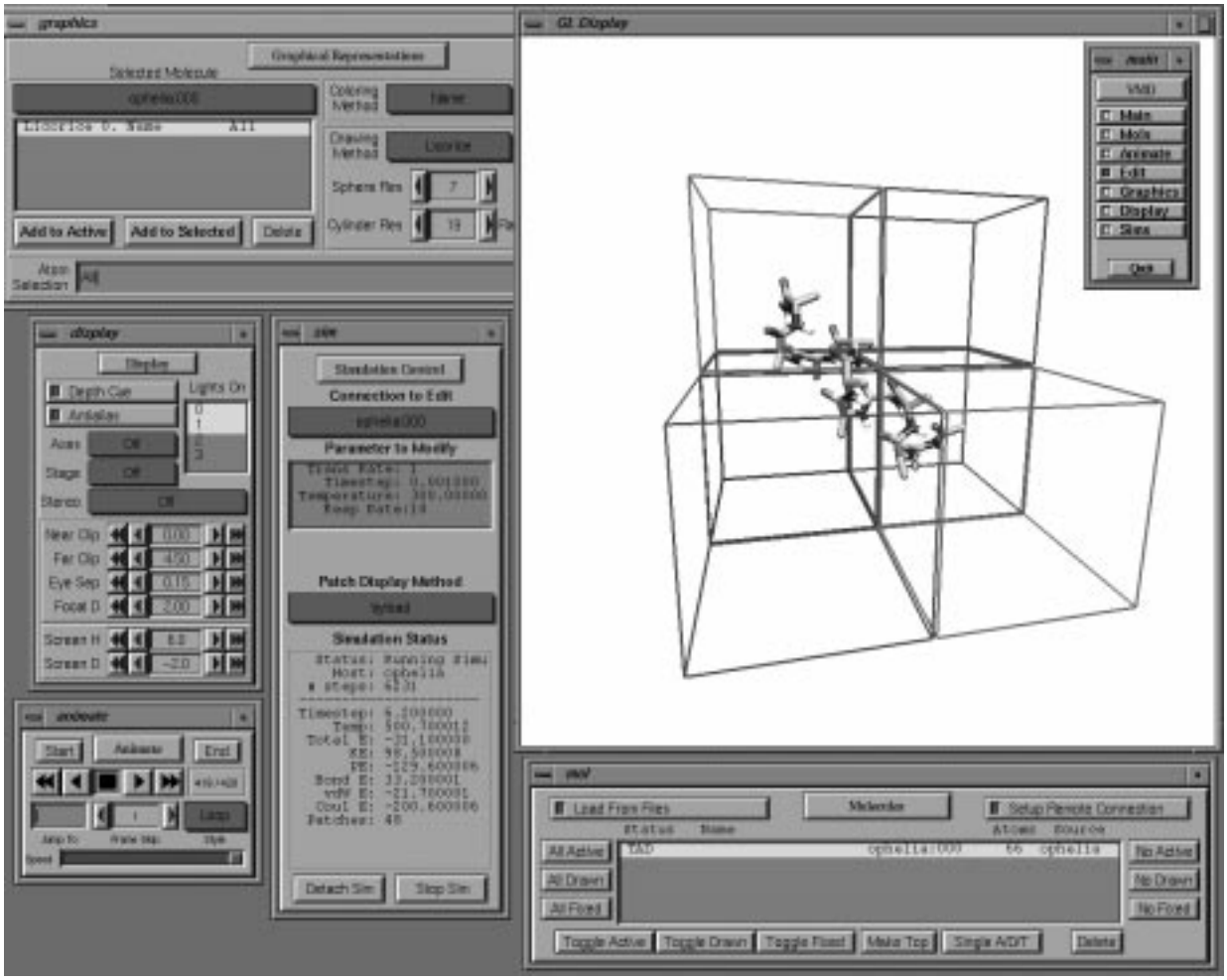

Figure 3:
Sample vmd session, showing a remote simulation connection and the use of the graphical user interface. After initializing the simulation as shown in fig. 4, vmd retrieves the molecular structure from the remote computer and then displays the results of the molecular dynamics calculations as they are computed. The MD application used here is namd, described in section 3; the boxes surrounding the polypeptide visualize the spatial decomposition used in the parallelization strategy of namd.

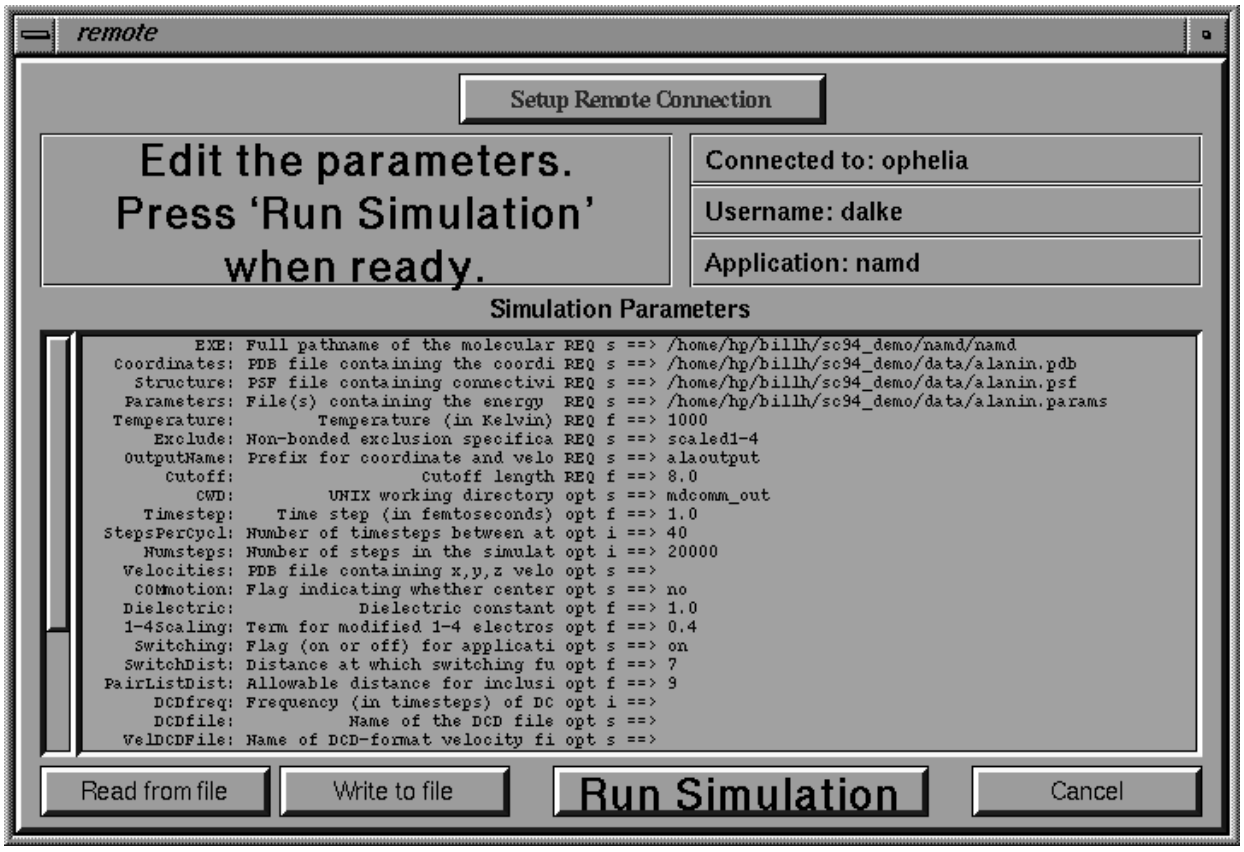

Figure 4:
vmd screen image illustrating the use of the graphical user interface to initiate a molecular dynamics simulation on a remote computer. Here a connection to the machine "ophelia" has been established, and the simulation options are being edited.

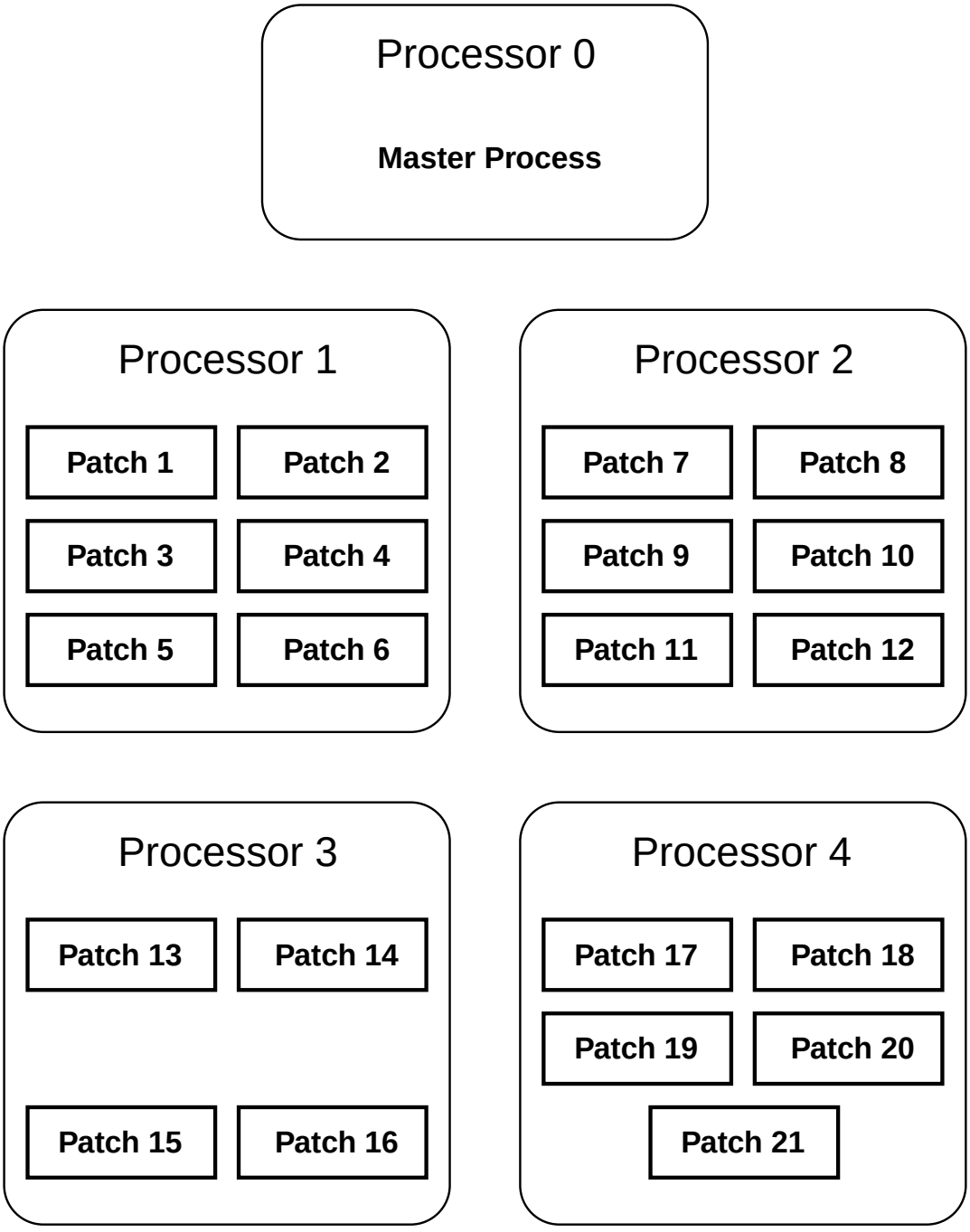

Figure 5:
Example mapping of patches to processors in namd. Multiple patches are mapped to each processor allowing for load balancing by reassigning patches to processors and for efficient scheduling of tasks on each processor to avoid idle processor time.

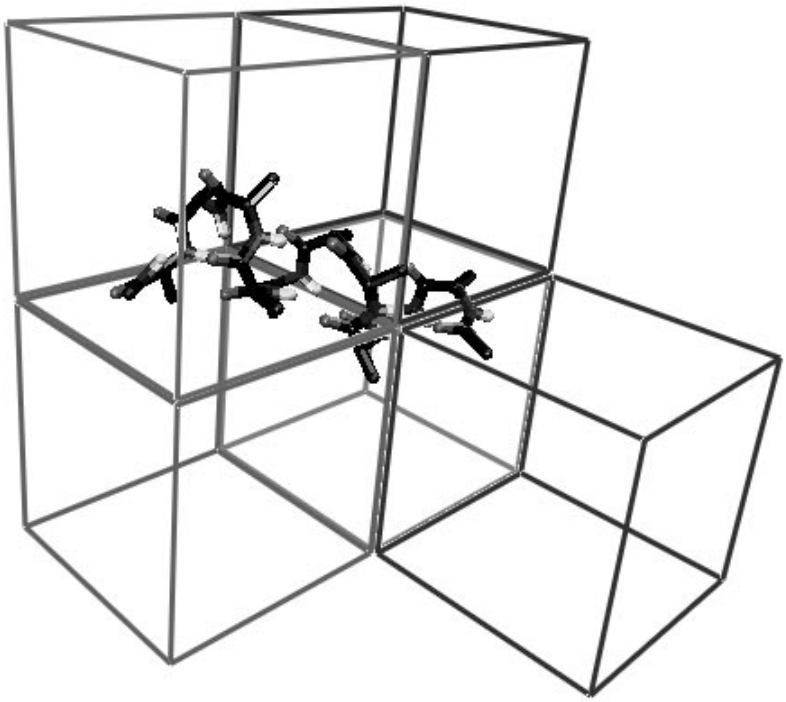

Figure 6:
Spatial decomposition of a small polypeptide. Each cube represents a patch in namd.

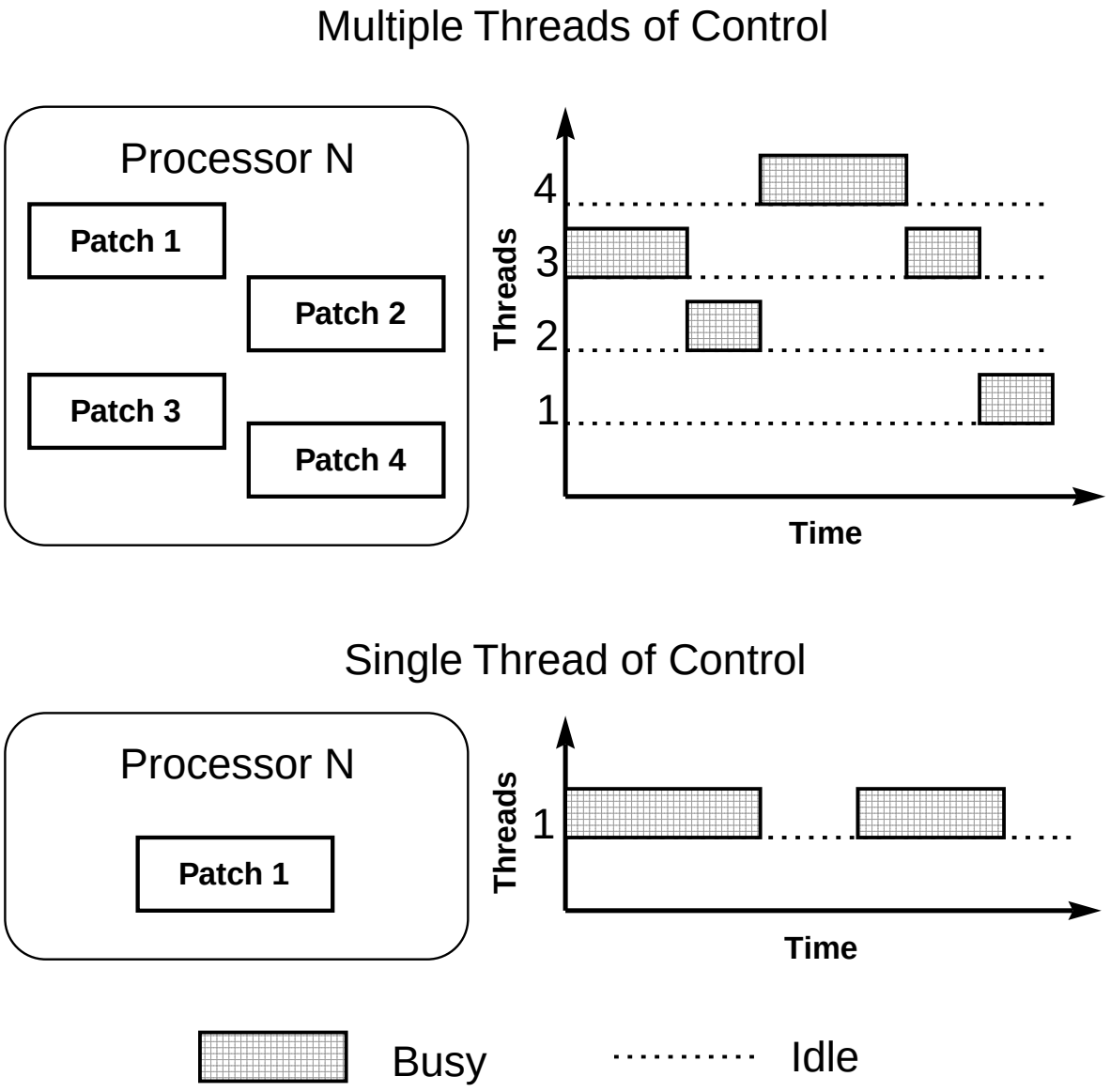

Figure 7:
Processor utilization comparison between single-threaded and multi-threaded execution. With the single-threaded execution, the processor is idle while waiting for data for the next computation. With the multi-threaded execution, when one thread is blocked, another thread is scheduled thus reducing the amount of idle processor time.

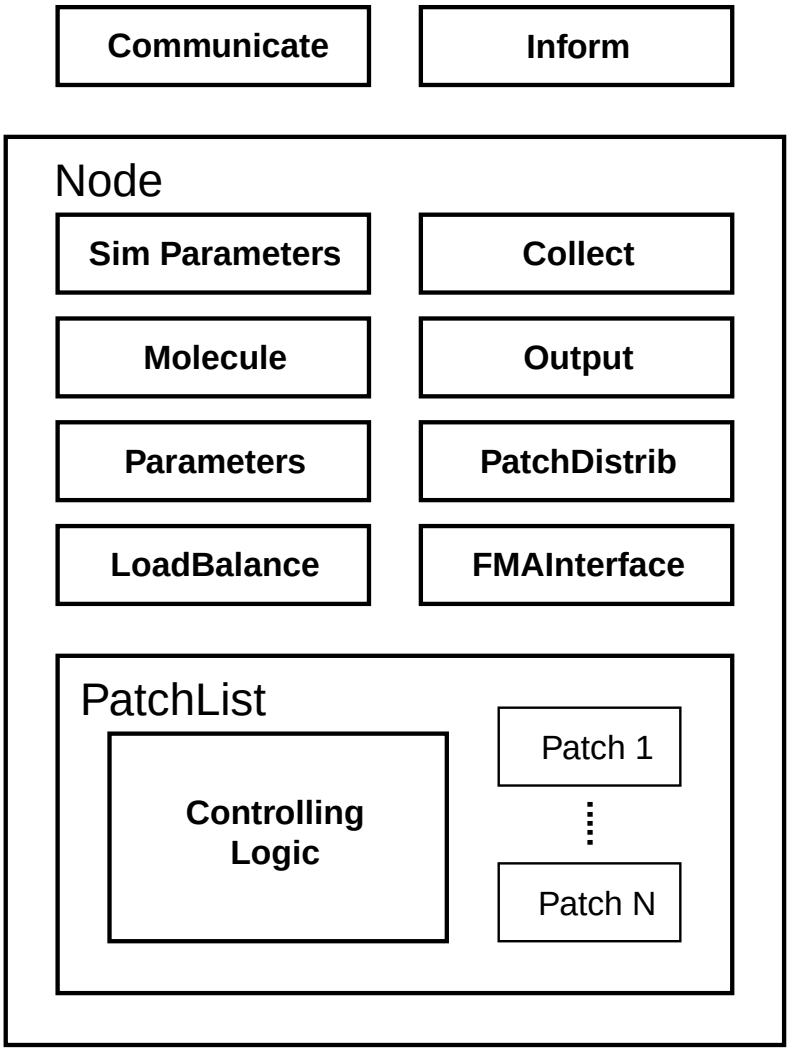

Figure 8:
Processor level object diagram for namd. Each of the objects shown exists on each processor running namd. A description of the purpose of each object is given in table 1

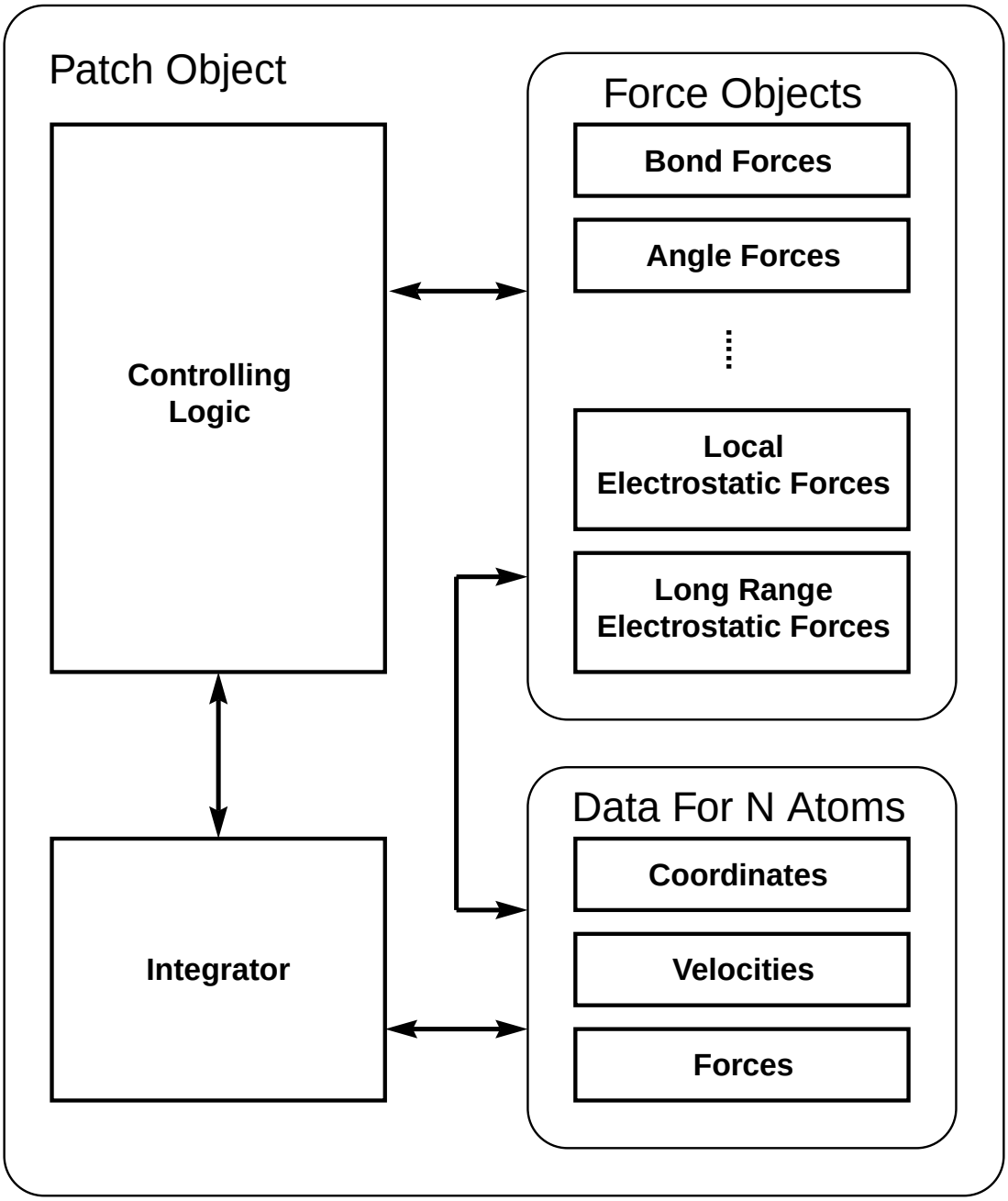

Figure 9:
Patch level object diagram for namd. Each patch consists of a data section which includes the coordinates, velocities, and forces for each atom, force objects which compute components of the force field, an object to perform integration, and the controlling logic that calls each object appropriately.

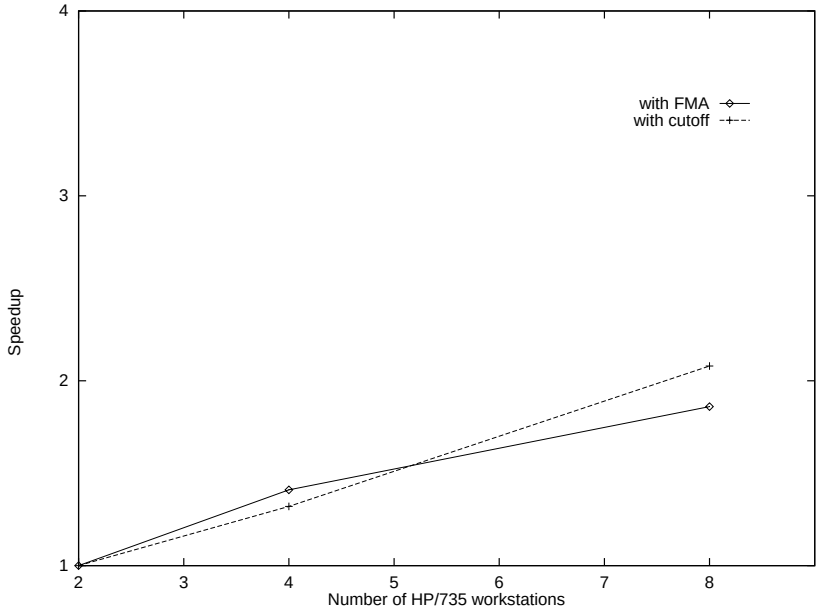

Figure 10:
Speedup of namd on a network of HP-735 workstations for the glucocorticoid receptor DNA binding domain (13,566 atoms altogether), with DPMTA and without DPMTA. The speedup reported is with respect to the two-processor performance. The simulations were done with an 8 Å cutoff distance; DPMTA calculations were carried out using 5 levels and 4 multipole terms.

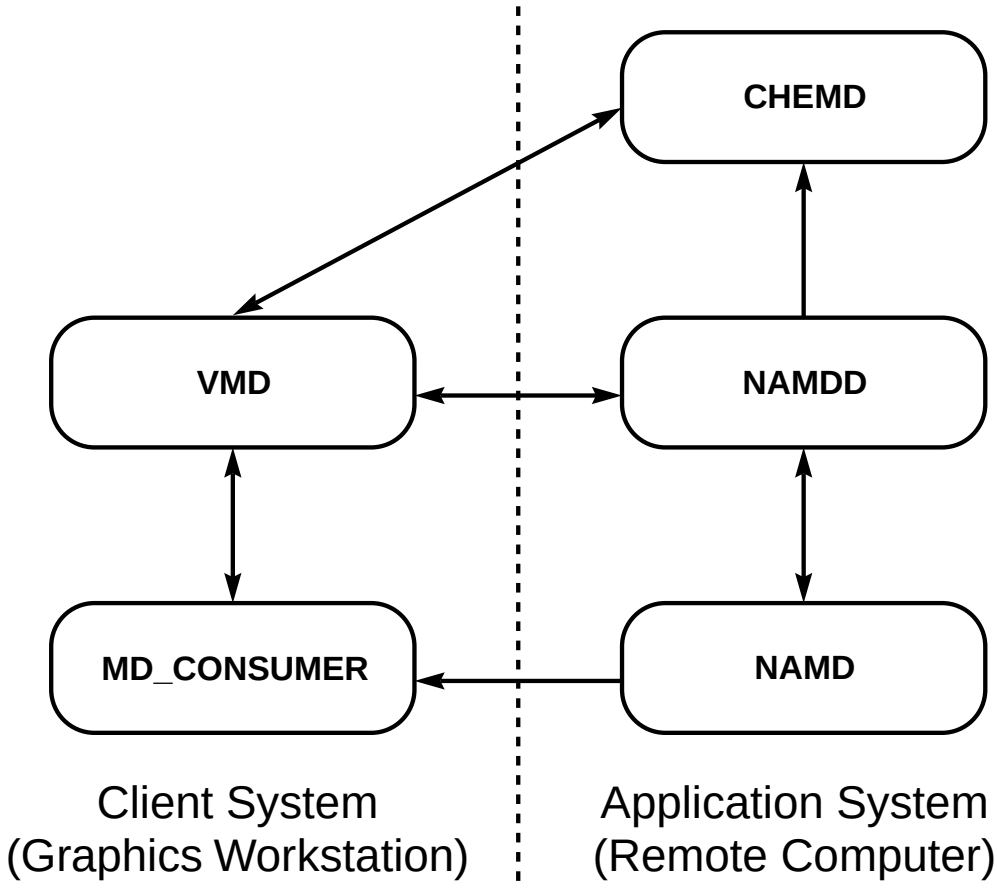

Figure 11:
MDCOMM coordination of namd and vmd. The application system may be the same local host on which a user is running vmd, or a remote computer.

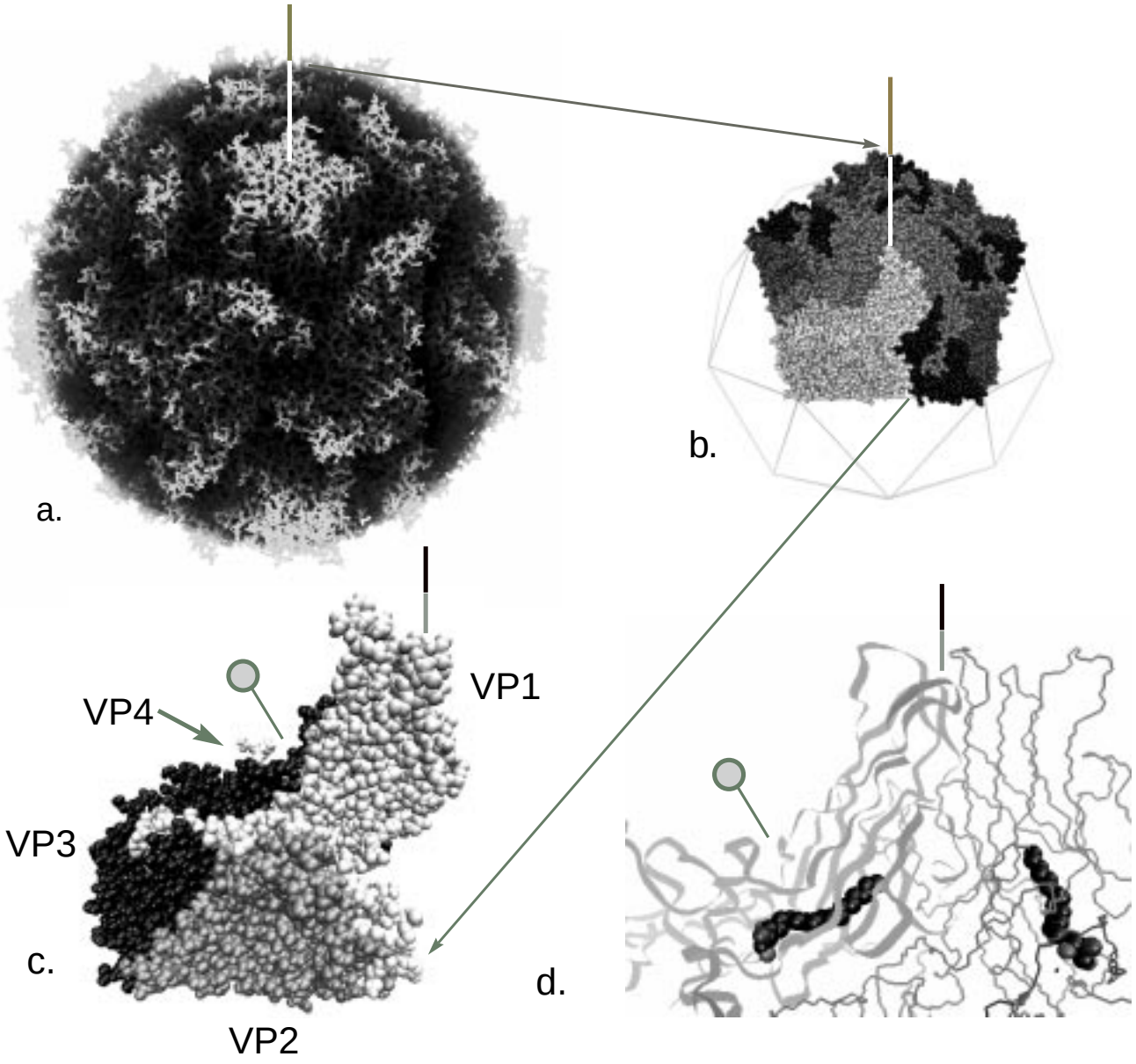

Figure 12:
The coat of poliovirus, shown radially depth-cued in (a), is naturally assembled from 12 copies of one pentamer, shown in (b). The pentamer is made of five copies of one protomer, which is itself comprised of four proteins as illustrated in (c). A sphingosine sits inside of the VP1 protein and is implicated in controlling the stability of the virus coat. The interface between two protomers is shown in (d), with one protomer rendered in ribbons and the other as a backbone trace. The sphingosines are shown as solid spheres. All figure elements were rendered using vmd.