STALK: An Interactive Virtual Molecular Docking System

David Levine* Michael Facello† Philip Hallstrom‡ Greg Reeder‡
Brian Walenz‡ Fred Stevens§

Abstract

Several recent technologies—genetic algorithms, parallel and distributed computing, virtual reality, and high-speed networking—provide the foundation for a new approach to the computational study of molecular interactions. Parallel genetic algorithms are an efficient and effective means to explore the large search spaces typical of these problems, while virtual reality provides an immersive environment for visualizing the interactions. In this paper we discuss the STALK system, which uses high-speed networking to couple a parallel genetic algorithm to a virtual reality environment. This combination allows a local or remote user to interact in real-time with the simulation through the virtual reality environment. Molecular docking experiments using an IBM SP parallel computer and a CAVE virtual reality environment are discussed.

1 Introduction

A major challenge for computational biology is the development of efficient algorithms for determining the sites of interactions between macromolecules and other macromolecules or small ligands. These interactions include protein-drug interactions, protein-protein interactions such as found in antibody-antigen complex formation, and interactions of nucleic acids with proteins and other ligands.

The most important application is in the pharmaceutical industry, in which the ability to computationally screen potential interactions between hypothetical ligands and receptors could markedly speed the development of new drugs, reduce the cost of development, and improve efficacy. As conformations of biomedically relevant proteins continue to accumulate at an accelerated pace through crystallographic and nuclear magnetic resonance studies, the possible contribution of computational screening grows in parallel.

Contemporary computational tools can provide substantial insight into the mechanisms of interaction found in complexes observed crystallographically, as well as guidance into protein

*Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois 60439.
†Computer Science Department, University of Illinois, Champaign, Illinois 61801.
‡Science and Engineering Research Semester Program, Argonne National Laboratory, Argonne, Illinois 60439.
§Center for Mechanistic Biology and Biotechnology, Argonne National Laboratory, Argonne, Illinois 60439.

engineering approaches to optimize antibody-antigen interactions, or ligand modifications to improve drug binding by a receptor molecule. However, given only the three-dimensional structures of two molecules that interact through unidentified interfaces, the success of computational methods in predicting the existence of interaction and identifying the sites of interaction has been limited.

In part, the limited success of computational methods is a consequence of a requirement faced by all computational analyses of macromolecules: the necessity to use approximated force fields and solvent representations to accommodate finite computational resources and manageable execution times. Additionally, protein-protein interactions typically involve micro- to milli-second timeframes. Computational studies of these interactions are not feasible using traditional molecular dynamics approaches because of the overwhelming computational requirements associated with the femtosecond time scale required.

Several recent technologies—genetic algorithms, parallel and distributed computing, virtual reality, and high-speed networking—provide the foundation for a new approach to the computational study of molecular interactions. Using these techniques we have developed the STALK system for studying molecular docking. In this paper we discuss preliminary experiences with STALK.

2 Underlying Technologies

In this section we briefly discuss the key technologies upon which our molecular docking system is built.

2.1 Parallel Computing

The most popular parallel computing architecture today is the distributed-memory multiple instruction multiple data (MIMD) computer. These systems have multiple "nodes." A node consists of a processor and memory, a network interface, and usually a local disk. The nodes are connected via a network that allows them to communicate with each other. Parallelism is achieved when the processors compute simultaneously on the data in their memories. Both large-scale massively parallel processors (MPPs) and workstation networks are members of this architecture class.

Message passing is a natural programming model for distributed-memory MIMD computers. In this model, software processes are mapped onto the computer's nodes and communicate by passing messages—transferring data from the address space of one process into the address space of another process. The Message Passing Interface (MPI) is a recently defined standard for message-passing [5, 8]. MPI defines a set of functions, parameters, and their behavior. MPI was designed by a large group of parallel-computer vendors, computer researchers, and application developers as a standard for message passing. Both vendor-specific and third-party versions of MPI exist.

The master/slave model is a parallel programming model that can be naturally implemented using message passing. In this model, a *master* process distributes work (computations to be

performed) to slave processes. The slaves perform the work and return the result to the master. In many implementations, the master plays a bookkeeping role only and does not perform any computation.

2.2 Wide-Area Distributed Computing

The goal in wide-area distributed computing is to integrate geographically-distributed, heterogeneous computing resources, such as parallel supercomputers, database servers, and graphics environments, via high-speed networks into a unified, distance-independent environment. Many people believe that in the current fiscal environment this is a more realistic scenario for increasing computing power than is trying to situate all of these resources in one place. However, such environments raise a number of issues.

Systems issues that arise include the need for a uniform authentication mechanism, distributed process startup, and a single interface to schedule and initiate runs. Software issues include how to identify the different computational resources, how to handle different data representations on these resources, and how to develop and maintain applications across the resources. The main performance issues are optimizing communication routines, for example, allowing communication inside a resource to use resource specific communication, rather than a general resource to resource method, and developing load balancing techniques, since resources will typically contain different amounts of processing power.

2.3 Genetic Algorithms

Genetic algorithms (GAs) are search algorithms. They were developed by Holland [10] and are based on an analogy with natural selection and population genetics. One popular use of GAs is for finding approximate solutions to difficult optimization problems. Unlike other optimization methods, genetic algorithms work with a *population* of candidate solutions instead of just a single solution. We use the term "string" to refer to an individual solution.

Genetic algorithms work by assigning a value to each string in the population according to a problem-specific fitness function. A "survival-of-the fittest" step selects strings from the *old* population, according to their fitness. These strings recombine using operators such as crossover (swapping substrings) and mutation (random perturbations of a string's value(s)) to produce a new generation of strings that are (one hopes) more fit then the previous one. These new strings are than evaluated by the problem-specific fitness function. A generic genetic algorithm is shown in Figure 1.

Although most of the steps of a genetic algorithm can be executed in parallel, in many real-life applications, including ours, the evaluation of a string is the dominant cost. In these cases, a master/slave model where the master process distributes strings to the slave processes for evaluation will usually be computationally efficient. The PGAPack [14, 15] parallel genetic algorithm library provides a parallel master/slave implementation and was used in our work.

PGAPack is a general-purpose, data-structure-neutral, parallel genetic algorithm library. It provides most genetic algorithm capabilities and features in an integrated, seamless, and portable

```
t ← 0
initialize P(t)
evaluate P(t)
foreach generation
    t ← t + 1
    select P(t + 1) from P(t)
    recombine P(t + 1)
    evaluate P(t + 1)
endfor
```

Figure 1: Simple Genetic Algorithm

manner. Key features include C and Fortran interfaces, binary-, integer-, real-, and character-valued native data types, object-oriented design, multiple choices for GA operators and parameters, and easy extensibility. PGAPack makes MPI message-passing calls and so can run on most parallel computers and workstation networks.

2.4 Virtual Reality

A virtual reality (VR) system uses visual and audio cues such as wide field of view, stereo display, viewer-centered perspective, and localized and synthesized sound to provide an *immersive* environment for the user. Additionally, some VR systems, by tracking the user's instantaneous position and orientation and allowing the user to manipulate objects, provide an *interactive* environment. In this way, virtual reality strives to be a more natural user interface. It allows the scientist to focus on the data rather than the computer interface and to take advantage of the human ability to process 3-D spatial information [1, 13].

The CAVE[1] (CAVE Automatic Virtual Environment) [4] is the virtual reality system we used. The CAVE is a $10 \times 10 \times 9$-foot cube in which the user is surrounded by stereoscopic computer images rendered on the walls and floor. Left- and right-eye images are projected onto the walls and floor in rapid alternating succession to create a 3D stereo effect. A person standing inside the CAVE wears LCD shutter glasses that synchronize the left and right eye views, giving the illusion of three-dimensional immersion. The user is tracked by an electromagnetic tracking system, so that his or her instantaneous position and orientation are known. Figure 2 is a picture of the CAVE environment.

The user is able to manipulate objects within the CAVE by using a wand, a three-dimensional analog of a computer mouse. The wand has three buttons and provides a joystick interface to the CAVE simulation. The position and orientation of the wand is also tracked by the electromagnetic tracking system. The CAVE allows multiple users to share the virtual experience by each wearing LCD shutter glasses.

The CAVE is driven by a Silicon Graphics (SGI) Onyx workstation. The SGI Onyx is a

[1] The Cave is a trademark of the Board of Trustees at the University of Illinois.

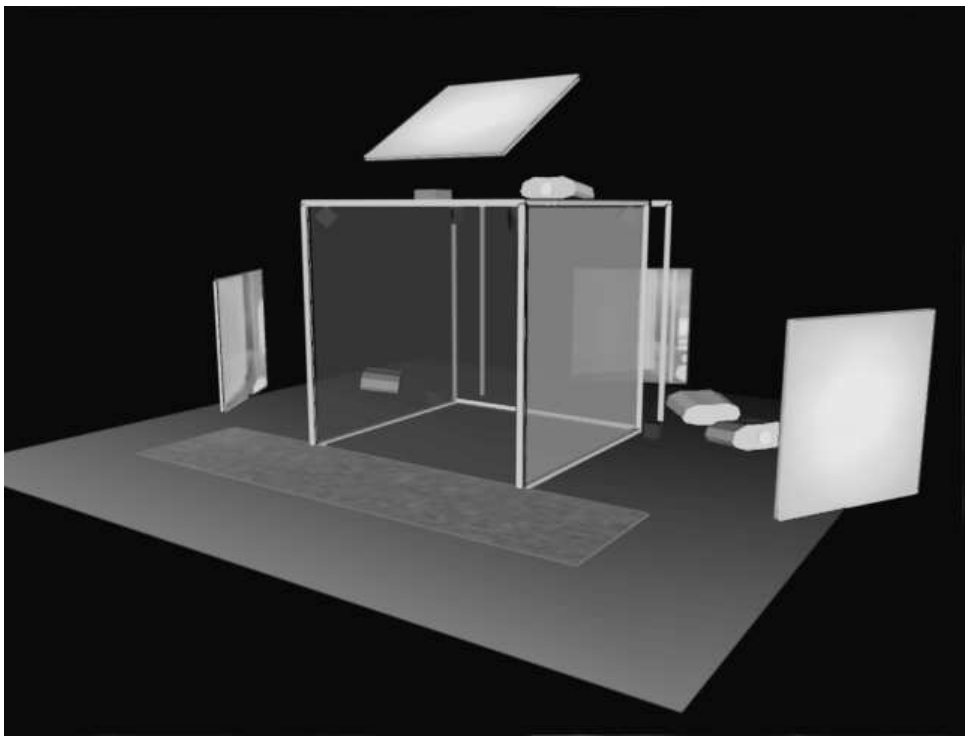

Figure 2: The CAVE virtual reality environment. Computer images sent to the projectors are directed onto the CAVE walls and floor.

shared-memory multiprocessor. The graphics subsystem for the Onyx at Argonne National Laboratory has 256 MB RAM, 10 GB disk, four R4400 processors, and three RealityEngine^2 graphics coprocessors. Each graphics coprocessor is connected to a high-resolution projector that projects the image onto the walls of the CAVE.

All of the CAVE code is executed on the SGI Onyx. CAVE applications are typically written in C or C++ and make OpenGL or OpenInventor calls. Calls to the CAVE library manage the computation of user-centered perspective, synchronization of frames across the walls, and tracking and wand I/O.

3 Molecular Docking Formulation

Computational methods for the evaluation of protein-ligand and protein-protein interactions have received substantial attention in recent years (see, for instance, [9, 11, 16, 17, 18, 21] and references cited therein). This attention is prompted both by the fundamental and applied significance of these interactions, and by the improvements in computational resources that have made such methods feasible.

As described by Kuntz and coworkers [16, 17, 18, 19], docking methods can be characterized by three different strategies by which the interacting molecules are juxtapositioned. First, the ligand may be positioned manually, guided by some experimental data that provides clues to the identity of the interacting surface and to the complex evaluated by energy minimization. In

a second approach, the ligand is positioned in an approximate location, and molecular dynamics used to bring the molecules together in an energetically favorable relationship. As in the first strategy, substantial experimental guidance is required to make this practicable. Finally, in other cases, little a priori information about the geometry of the interaction is necessary, and exhaustive searches and evaluations of potential complex formations are made. Success is highly dependent on the thoroughness of the search strategy [16]. The effectiveness of genetic algorithms as search optimization techniques has prompted recent interest in its application to docking issues [2, 7, 12, 22].

Our goal is to study protein-ligand docking. Numerically, this is formulated as an optimization problem where the goal is to minimize the free energy of the molecular system by maximizing the intermolecular interaction energy between the two molecules. In our formulation the protein molecule is fixed and the degrees of freedom are the translation and rotation of the center of mass of the ligand molecule. This yields six variables: three translation values and three rotation values. In this "rigid body" formulation, it is assumed that no modification of the protein backbone or side chain positions occurs.[2] The goal of the optimization procedure is to search the space of possible conformations for the lowest energy configuration between the two molecules.

The energy, E, is computed by using a Lennard-Jones potential energy function for Van der Waals energy terms and Coulombic term for electrostatic contributions [16]. We define the following terms. Let P_i be the set of atoms in molecule i (the protein or the ligand), with $|P_i| = n_i$. Let a_{ij} be the jth atom of P_i. Let q_{ij} be the charge of atom j in molecule i. Let $d(a_{ij},\ a_{i'j'})$ be the Euclidean distance between the two atoms. Let D be the dielectric constant and A_{ij} and B_{ij} be Van der Waals constants dependent on the type of atom. Then the energy is given by

$$\begin{aligned} E \;=\; & \sum_{j=1}^{n_1}\sum_{j'=1}^{n_2} A_{1j}\ A_{2j'} \ /\ d(a_{1j},\ a_{2j'})^6 - B_{1j}\ B_{2j'} \ /\ d(a_{1j},\ a_{2j'})^{12} \ + \\ & \sum_{j=1}^{n_1}\sum_{j'=1}^{n_2} 0.322\ q_{1i}\ q_{2i'} \ /\ (D\ d(a_{1j},\ a_{2j'})). \end{aligned} \tag{1}$$

This formula implies an $O(n_1 n_2)$ time algorithm for computing E, which can be computationally expensive for large macromolecules. To reduce the computation time, we approximate E. We do this by using a three-dimensional subdivision of the space, with the size of a cell of the subdivision specified by a parameter. The cell containing an atom is computed for each atom. When E is computed, only pairs of atoms that lie in the same cell or immediately adjacent cells, contribute to the energy sum. For the results presented later in this paper, a cell size of 10 Å was used.

Each string in the GA population represents a potential minimizer of Eq. (1). Each string has six parameters. The first three parameters represent the x-, y- and z-translations of the ligand. The second three parameters represent the x-, y-, and z-rotations about the center of mass of the protein. Each parameter is represented by using a floating-point number.

When a run is made, the center of mass of the protein is translated to the origin of the system and remains stationary throughout the run. The GA strings are generated randomly. The initial

[2] Allowing sidechain angles as additional degrees of freedom is under development.

translation values are selected uniformly randomly from a parameterized box about the protein center of mass. The rotation values are selected uniformly randomly from the range $[-\pi, \pi]$. From the six parameters and the initial coordinates of the ligand atoms, one can compute a new set of atom centers using linear transformations of the atom coordinates by using matrix operations.

The formulation used allows for the possibility the molecules may overlap. The traditional approach in these situations is to define a penalty term to add to Eq. (1) to degrade the fitness of the string when an overlap occurs. Although we experimented with explicit penalty terms, we found that since the Van der Waals component of the energy is very high when atoms are close together, this provides a natural overlap penalty, and therefore we do not need an explicit penalty.

4 STALK Architecture and Features

The system we developed for studying protein-ligand docking is named STALK. STALK consists of two parts: a numerical simulation program and a visualization program. The numerical simulation program runs the genetic algorithm on a parallel computer using the master/slave model to execute the function evaluations in parallel. It is written in Fortran and makes PGAPack and MPI library calls. The program's main components are routines to run the genetic algorithm, evaluate Eq. (1), and communicate with the visualization program.

The visualization program communicates with the numerical simulation program allowing both observation of the genetic algorithm's progress and an interactive interface. The visualization program is written in C and makes OpenGL and CAVE library calls. The visualization program is a *shared*-memory program with one process for wand navigation, one process for computation, and three processes for display. These processes communicate with each other through a set of shared variables. The navigate process polls the wand and manipulates the program based on the program states. The computation process handles all communication with the numerical simulation program. Each display process updates the view on one wall of the CAVE.

Each iteration of the genetic algorithm, the string corresponding to the lowest energy conformation found so far is transmitted to the CAVE. From the string's parameter values the visualization program computes and displays the low-energy conformation in the CAVE. The molecules are drawn as a collection of spheres that represent the different atoms, connected by lines representing molecular bonds. The current GA generation, the string's parameter values, and the energy are displayed on the front wall of the CAVE. This capability allows the user to monitor the progress of the GA and to examine the best conformation found. Figure 3 is a view from the CAVE simulator showing the output of the visualization program.

Several features allow the user to modify the CAVE display. Using the wand, the user can translate or rotate the conformation displayed. A menu displayed on one of the CAVE walls supports additional display features. One is a toggle that allows the ligand atoms to be displayed in cyan in order to differentiate them from the protein atoms. A second menu feature is a surface representation display of the protein molecule. A third menu feature removes the sidechain atoms from the display and displays only the backbone atoms. Finally, the atoms in each sidechain

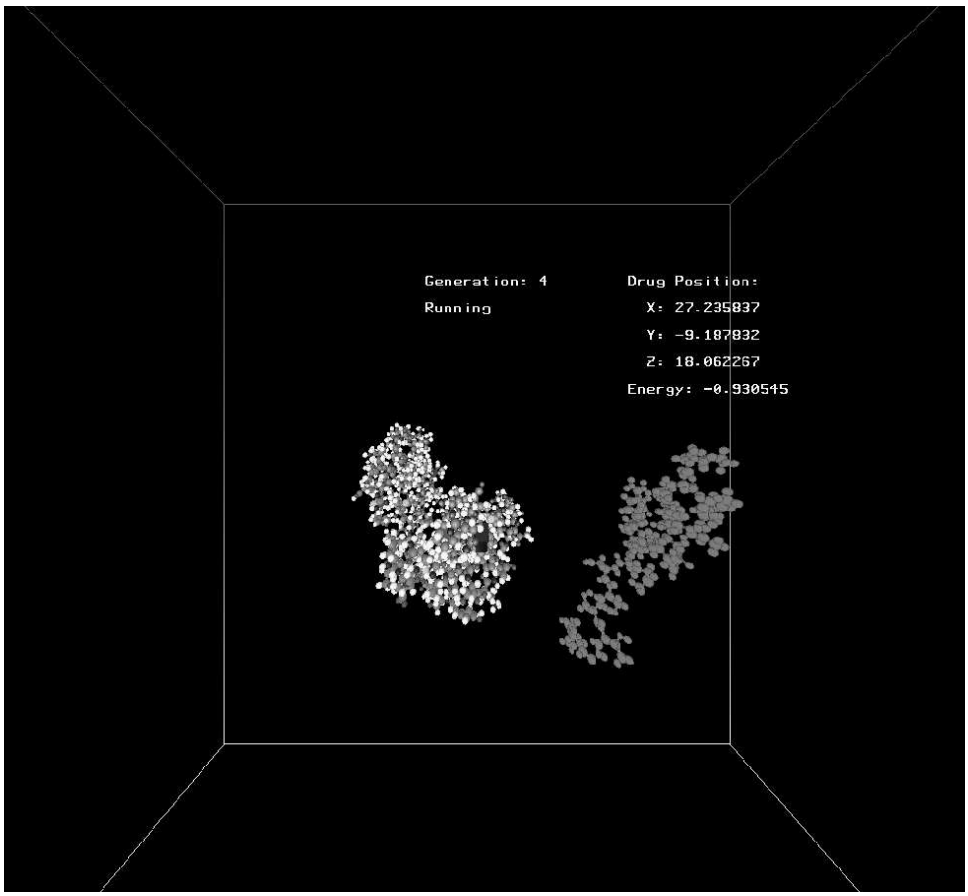

Figure 3: Output of the Visualization Program Displayed in the CAVE Simulator.

can be highlighted by displaying them in orange.

The visualization program allows the user to interact through the CAVE environment with the numerical simulation program in real time. To do this, the user first selects a menu item to suspend the genetic algorithm. Using the wand, the user may translate or rotate the ligand relative to the position of the protein. The visualization program communicates the ligand's new position and orientation to the numerical simulation program. The user may either request an energy evaluation or restart the GA. If an energy evaluation is requested, it is performed and the new energy value returned. If the GA is restarted, the new ligand position may either be ignored or be incorporated into the numerical simulation in one of two ways: the new ligand parameters can be used to replace the worst population member, or, alternatively, the current GA population can be replaced with strings that are random perturbations of the new ligand parameters.

This interactive capability allows scientists to use their intuition to assist the algorithm in finding a low-energy conformation, or to specify alternative starting positions from which to study the docking process. A recent effort, similar in spirit to STALK, is VIBE [3]. VIBE couples a CAVE and IBM SP together in the context of a molecular dynamics algorithm.

5 Results

In this section we discuss our early experiences with STALK. We performed the experiments in Section 5.1 to measure the efficiency and accuracy of the numerical simulation program. Section 5.2 is more qualitative and discusses our experiences using the interactive capabilities of STALK.

5.1 Numerical Results

The test case we used is the protein Ribonuclease S. Ribonuclease S is formed by the cleavage of the peptide bond between positions 20 and 21 of Ribonuclease A. We used the helix-forming S-peptide as a model ligand, and the S-protein as a model receptor. The S-peptide has 297 atoms, and the S-protein has 1,564 atoms. The coordinates for Ribonuclease S are taken from the Brookhaven Database entry, 1RBH [20].

Applying the potential of Eq. (1) to the coordinates from [20], an energy of -37.8 kcal/mol was calculated. To test the GA we randomly translated and rotated the S-peptide away from the S-protein and then used the numerical simulation program to attempt a redocking to the crystallographically determined position. Our experiments were performed on an IBM SP parallel computer with 128 nodes, each of which consisted of an IBM RS/6000 Model 370 workstation processor, 128 MB of memory, and a 1 GB local disk.

Each test run used a unique random number seed. The following GA parameters were used. The population size was 1,000 and 100 strings were replaced each GA iteration. New strings were created either via uniform crossover (with probability 0.9) *or* mutation (with probability 0.1). Mutation was performed by randomly adding or subtracting a quantity generated from a Gaussian distribution with mean 0.0 and standard deviation 0.1. The mutation rate was 1/6.

Two sets of runs were made. In the first, the translation values were randomly initialized by using a 2 Å box, and the rotation values were in the range $[-\pi, \pi]$. For each of 1, 2, 4, 7, 14, 26, 51, and 101 processors, six runs of 4,000 iterations each were made. In the second set of runs, the translation values were randomly initialized by using a 100 Å box, and the rotation values were in the range $[-\pi, \pi]$. The same number of total runs were made.

The results of the runs are summarized in Table 1. The first column is the box size used for generating translation values. The second and third columns are the mean energy and associated standard deviation from all runs. The fourth and fifth columns are the minimum and maximum energy values found. Although the means are approximately equal, the standard deviation is significantly higher in the case of the 100 Å box. The wider variability results in significantly better, and significantly worse, solutions being found.

Table 1: Docking Statistics

Box Size	Mean Energy	σ Energy	Min. Energy	Max. Energy
2 Å	-41.8	2.2	-44.2	-36.9
100 Å	-42.0	5.5	-58.2	-32.7

Performance results are given in Table 2 for the 2 Å case. The *Total Proc.* column is the number of SP processors used. The *Compute Proc.* column is the number of processors that execute function evaluations. For the special case of exactly two processors, both the master and slave processes perform function evaluations. When more than two processors are used, one processor runs the master process, and the other processors perform function evaluations. The *Time* column is the average over six runs of the total time spent by the master process (executing the GA, packing and sending data to the slave processes, and waiting for results). The *Speedup* column is the ratio of the time to execute the one-processor case to the time to execute with that number of processors. The speedup achieved is fairly constant, although not ideal.

Table 2: Solution Time vs. Number of Processors

Total Proc.	Compute Proc.	Time (sec.)	Speedup
1	1	263581	1.0
2	2	148666	1.8
4	3	87208	3.0
7	6	46950	5.6
14	13	22150	11.9
26	25	12831	20.5
51	50	7193	36.6
101	100	4181	63.0

5.2 Interactive Usage

We developed STALK using the environment shown in Figure 4. This figure shows the 128-node IBM SP that ran the numerical simulation program, the SGI Onyx that ran the visualization program, and the CAVE system. The network can be configured to use Ethernet, ATM, or HIPPI. During our development we used the HIPPI connection which runs at 800 Mbps. One problem we faced was that the IBM SP was part of a production environment, whereas the SGI/CAVE was part of an experimental environment. In particular, issues pertaining to security, program startup, and filesystems had not been previously addressed and required significant effort on our part.

To test the computational steering capabilities, we let the GA run for an arbitrary, but small (usually less than 100), number of iterations, suspended the numerical simulation program, and attempted to dock the ligand using the wand. The mechanics of "hand-docking" consisted of stepping into the CAVE and using the wand to translate and rotate the ligand to try to fit it into the cleft in the protein. Once this was completed to the user's satisfaction, the new ligand coordinates were sent to the numerical simulation program which returned an energy evaluation. Figure 5 shows a user inside the CAVE environment wearing LCD shutter glasses and holding the wand.

In general, we found the hand-docked solutions were not as good as those the GA had already found. One limitation was a significant lag from when the user moved the wand until the updated position was shown in the CAVE. This was due to the large number of spheres that needed to be redrawn, and the slow sampling rate of the wand tracker. The result was that the user would

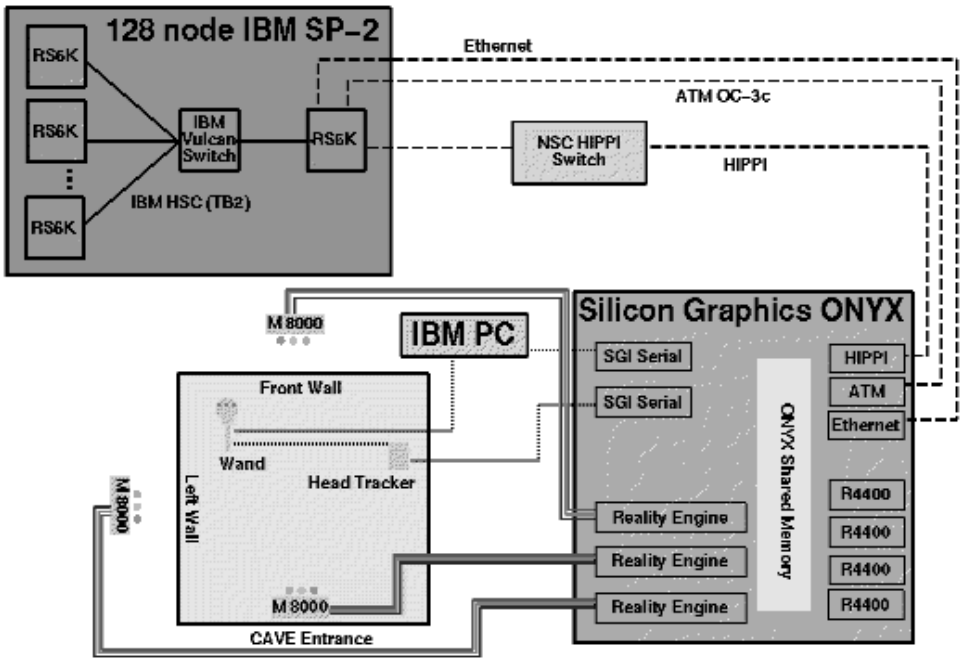

Figure 4: STALK Development Environment.

move the wand based on the rendered position of the ligand, not on the position last read by the tracker. This led to jerky motion that made an accurate hand-docking difficult to achieve.

A second limitation was one of context. When immersed in a conformation, the large number of atoms made it difficult to determine which part of the molecule one was viewing. We found it necessary to view the visualization from the front, rather than immersively, in order to maintain a sense of perspective. The disadvantage of this, however, is that we did not take advantage of the immersive capabilities, but instead tried to hand-dock the ligand from "afar." Using the option to remove sidechain atoms from the display helped alleviate the context problem. However, since the sidechain atoms were not visible it introduced another problem—an inability to determine if there were overlapping atoms in the new conformations.

Since the hand-docked solutions were invariably worse than those already in the population, replacing the worse population member with the hand-docked solution had no effect. Regenerating the entire population by perturbing the hand-docked solution invariably made the best solution found so far worse. However, the robustness of the GA usually found a "good" solution again quickly, although not any better, it appeared, than if the GA had run uninterrupted.

We tested wide-area usage of STALK at the Supercomputing '95 conference as part of the Information Wide Area Year (I-WAY) project. The I-WAY was an experimental high-speed network that linked together a number of high-performance computers and advanced visualization environments around the country. Two important components of the I-WAY project were the backbone network and the software environment.

The backbone I-WAY network was based on Asynchronous Transfer Mode (ATM) technology. ATM is an emerging standard for advanced telecommunications networks. It is capable of simultaneously carrying voice, data, and video significantly faster than conventional wide-area

Figure 5: A User in the CAVE.

networks. Much of the I-WAY consisted of existing fiber-optic lines that commercial long-distance carriers already had in place. The I-WAY network supported the Internet protocol (IP) over ATM.

One role of the I-Soft [6] I-WAY software environment was to schedule the I-WAY computational resources. In our case, these resources consisted of nodes on the IBM SP at Argonne National Laboratory (Illinois), nodes on the IBM SP in the Cornell Theory Center (New York), and the SGI Onyx/CAVE VR environment at Supercomputing '95 (San Diego). A second role was process creation and communication. For us, this required starting the numerical simulation program on the IBM SP(s) and the visualization program on the SGI Onyx. A third role was to handle and support different communication protocols for the MPI message-passing calls. For STALK, this meant that MPI calls within the IBM SP used a more efficient communication mechanism than the socket-based communication used to support MPI calls between the IBM SP and the SGI Onyx.

We performed two experiments in the I-WAY environment. In the first, we ran the numerical simulation program on 16 nodes of the IBM SP at Argonne National Laboratory and 16 nodes of the IBM SP at the Cornell Theory Center. Qualitatively, watching the GA iteration log being written at our workstation we did not observe any noticeable degradation in performance compared to similar runs that had been made on 32 nodes of a single IBM SP in a local environment. It is possible, however, that the computationally expensive function evaluations masked the network latency.

In our second experiment, we ran the numerical simulation program on 32 nodes of the IBM SP at Argonne National Laboratory, and the visualization program on the SGI Onyx in San Diego. The actual experiment consisted of running the Ribonuclease test case for groups of visitors, stopping the simulation, "docking" the ligand by hand, returning an energy evaluation, and resuming the simulation. This experiment ran continuously for approximately forty-five minutes and did not experience any hardware, software, or network failures.

6 Conclusions

Distributed computing systems that couple virtual reality front-ends to (possibly remote) MPP supercomputers over high-speed networks hold a great deal of potential for molecular scientists. The advantages of a virtual reality front-end are twofold. First, it allows a scientist to gain a deeper understanding of molecular docking by allowing immersive visualization of the conformations in three dimensions. Second, a scientist can use personal intuition to steer a simulation towards a lower-energy conformation or to specify an alternative starting position from which to study the docking process. The goal of such a system is to enable the discovery of solutions that are more effective than could be obtained by either the computer or the researcher alone.

Our experiences with STALK, however, show that several limitations must be overcome before such a system is useful in practice. One limitation was the lag from when the user moved the wand in the CAVE until the updated position was displayed. The lag led to situations where the user would "oversteer" because the rendered position of the ligand was not the position last read by the wand tracker. A second limitation was that the large number of atoms in the molecules made it difficult to determine which part of the molecule one was viewing. We found it necessary

to view the visualization from a distance, rather than immersively, in order to maintain a sense of perspective. In almost all cases we found that the hand-docked solutions were worse than those found by the GA.

Within the limitations faced by any docking method, the parallel genetic algorithm approach shows considerable promise. Many of the solutions found by the GA had energy values less than the crystallographically determined position. We found that initializing the GA in a region close to the crystallographically determined position led to more consistent results than initialization within a wider region, but that the overall best (and worst) solutions found were from those runs where the initialized region was not restricted.

We found that due to the complexity of Eq. (1), and the large number of strings that must evaluate this equation, the GA approach is very computationally expensive. Further, the computational expense can be expected to increase significantly when more realistic models (sidechain rotations, solvent representation) are used to model larger macromolecules. We found a parallel implementation was a necessity to solve our test problem in a reasonable amount of time.

Several avenues for future work exist. First, we are currently working on allowing sidechain rotations as additional degrees of freedom and would also like to include a solvent model. Algorithmically, we would like to combine the GA with a traditional energy minimization system to obtain more rapid convergence to a proposed docking solution, while avoiding premature identification of local minima solutions. Finally, additional development and refinement of the virtual reality interface are needed to make this a robust tool for the study of real molecular systems.

Acknowledgments

This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, under Contract W-31-109-Eng-38 and the Office of Health and Environmental Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

References

[1] S. Bryson. Virtual reality in scientific visualization. *Communications of the ACM*, 39(5):62–71, 1996.

[2] K. Clark and Ajay. Flexible ligand docking without parameter adjustment across four ligand-receptor complexes. *J. Comp. Chem.*, 16:1210–1226, 1995.

[3] C. Cruz-Neira, R. Langley, and P. Bash. VIBE: A virtual biomolecular environment for interactive molecular modeling, 1996. To appear in *Computers & Chemistry*.

[4] C. Cruz-Neira, D. Sandin, and T. DeFanti. Surround-screen projection-based virtual reality: The design and implementation of the CAVE. In *ACM SIGGRAPH '93 Proceedings*, pages 135–142. Lawrence Erlbaum Associates, 1993.

[5] Message Passing Interface Forum. MPI: A message-passing interface standard. *International J. Supercomp. Appls.*, 8(3/4), 1994.

[6] I. Foster, J. Geisler, B. Nickless, W. Smith, and S. Tuecke. *Software Infrastructure for the I-WAY High-Performance Distributed Computing Experiment.* In *Proceeeding of the Fifth IEEE Symposium on High Performance Distributed Computing.* IEEE Computer Society Press, 1996.

[7] D. Gehlhaar, G. Verkhivker, P. Rejto, C. Sherman, D. Fogel, L. Fogel, and S. Freer. Molecular recognition of the inhibitor AG-1343 by HIV-1 protease: Conformationally flexible docking by evolutionary programming. *Chemistry & Biology*, 2:317–324, 1995.

[8] W. Gropp, E. Lusk, and A. Skjellum. *USING MPI Portable Parallel Programming with the Message-Passing Interface.* MIT Press, Cambridge, 1994.

[9] M. Helmer-Citterich and A. Tramontano. PUZZLE: A new method for automated protein docking based on surface shape complementarity. *J. Mol. Biol.*, 235:1021–1031, 1994.

[10] J. Holland. *Adaption in Natural and Artificial Systems.* MIT Press, Cambridge, 1992.

[11] R. Jackson and M. Sternberg. A continuum model for protein-protein interactions: Application to the docking problem. *J. Mol. Biol.*, 250:258–275, 1995.

[12] R. Judson, E. Jaeger, and A. Treasurywala. A genetic algorithm based method for docking flexible molecules. *Computers & Chemistry*, 308:191–206, 1994.

[13] R. Kalawsky. *The Science of Virtual Reality and Virtual Environments.* Addison-Wesley, New York, 1993.

[14] D. Levine. `PGAPack`, 1995. A public-domain parallel genetic algorithm library. Available by anonymous `ftp` from `ftp.mcs.anl.gov` in the directory `pub/pgapack`, file `pgapack.tar.Z`.

[15] D. Levine. *User's Guide to the PGAPack Parallel Genetic Algorithm Library.* Technical Report ANL-95/18, Argonne National Laboratory, Mathematics and Computer Science Division, June 23, 1995.

[16] E. Meng, D. Gschwend, J. Blaney, and I. Kuntz. Orientational sampling and rigid-body minimization in molecular docking. *Proteins: Struct., Funct., Gen*, 17:266–278, 1993.

[17] E. Meng, B. Shoichet, and I. Kuntz. Automated docking with grid-based energy evaluation. *J. Comp. Chem.*, 13:505–524, 1992.

[18] B. Shoichet, D. Bodian, and I. Kuntz. Molecular docking using shape descriptors. *J. Comp. Chem.*, 13:380–397, 1992.

[19] B. Shoichet and I. Kuntz. Matching chemistry and shape in molecular docking. *Protein Eng.*, 6:723–732, 1993.

[20] R. Varadarajan and F. Richards. Crystallographic structures of Ribonuclease S variants with nonpolar substitution at position 13: Packing and cavities. *Biochemistry*, 31:12315, 1992.

[21] P. Walls and M. Sternberg. New algorithm to model protein-protein recognition based on surface complementarity: Applications to antibody-antigen docking. *J. Mol. Biol.*, 228:277–297, 1992.

[22] Y. Xiao and D. Williams. Genetic algorithms for docking of Actinomycin D and Deoxyguanosine molecules with comparison to the crystal structure of Actinomycin D-Deoxyguanosine complex. *J. Phys. Chem*, 98:7191–7200, 1994.