# Parallel Computing in Biomedical Research

Robert L. Martino, Calvin A. Johnson, Edward B. Suh, Benes L. Trus,* Tieng K. Yap

Scalable parallel computer architectures provide the computational performance needed for advanced biomedical computing problems. The National Institutes of Health have developed a number of parallel algorithms and techniques useful in determining biological structure and function. These applications include processing electron micrographs to determine the three-dimensional structure of viruses, calculating the solvent-accessible surface area of proteins to help predict the three-dimensional conformation of these molecules from their primary structures, and searching for homologous DNA or amino acid sequences in large biological databases. Timing results demonstrate substantial performance improvements with parallel implementations compared with conventional sequential systems.

High-performance parallel computers provide the computational rates necessary for advanced biomedical computing problems. Biomedical scientists can greatly reduce the time it takes to complete computationally intensive tasks and take new approaches in processing their data. This advantage may allow the inclusion of more data in a calculation, the determination of a more accurate result, a reduction in the time needed to complete a long computation, or the implementation of a new algorithm or more realistic model.

With proper computer network connections and interactive user interface, parallel computing is readily available to biomedical researchers in the laboratory or clinic at the investigators' computer workstations. This availability makes it possible to include the powerful resource of parallel computing as another tool that can be used in the research process. In the next section, we give a number of examples where parallel computing is being applied to biomedical research at the National Institutes of Health (NIH). We describe three of these examples in detail to demonstrate how parallel computing has been used to solve important computational problems in biomedicine. As shown by our first example, the parallel computer has become part of the process of determining the structure of the herpes virus capsid in a structural biology laboratory by greatly reducing the time it takes to obtain three-dimensional (3D) reconstructions of images from electron micrographs. We also access the parallel computer from workstations in the NIH Clinical Center to facilitate clinical investigations. For example, we are using a parallel method to align multiple positron emission tomography (PET) images to study the progression of Alzheimer's disease. The alignment method reduced the time for registering two PET images from 6 hours to 10 min. Significant time reductions in the completion of a computational task can completely change the way a procedure is implemented in the laboratory or clinic.

The authors are with the Computational Bioscience and Engineering Laboratory, Division of Computer Research and Technology, National Institutes of Health, Bethesda, MD 20892, USA.

*Joint appointment with the Laboratory of Structural Biology, National Institute of Arthritis, Musculoskeletal and Skin Diseases, National Institutes of Health, Bethesda, MD 20892, USA.

## Computational Challenges in Biomedical Research

With many of the institutes and the Clinical Center at NIH, we are working on a variety of research areas of biomedical interest that can benefit from the application of parallel computing. We list in Table 1 a number of representative scientific problems with their associated computationally intensive tasks that can benefit from high-performance parallel computing. For some of these problems, parallel methods simply speed up the completion of a laborious task. This is the case with the multidimensional nuclear magnetic resonance (NMR) spectroscopy example in which we automate the assignment of the signals in the NMR spectrum to the atoms of the protein under study (*1*). An important issue for this type of application is the interactive availability of the parallel computer on demand so that the user can appreciate its benefits. In other examples, parallel computing aids in the implementation of a complex theoretical model such as in a quantum chemistry calculation or molecular dynamics simulation. Here the researcher needs the system for extended periods of time. Recovery mechanisms must be in place in the event of a system failure during a long calculation or simulation.

Positron emission tomography is the most promising tool for biochemical imaging today. A PET image is formed through a computational reconstruction process. The quality of the resulting image and the computation time needed to produce it depend on the chosen reconstruction algorithm. Traditionally, Fourier methods such as the filtered back-projection algorithm, which is fast but can lead to artifacts, have been used. Another class of methods known as algebraic methods (*2*), an example of which is the expectation maximization (EM) or maximum likelihood (ML) algorithm, are known in theory to yield more accurate reconstructions or equivalent reconstructions with a lower patient dose. The algebraic methods have not been used in the past because of the long computation time and the large amount of memory required to implement them. In addition, a new generation of PET scanners allows for the retraction of the lead septa shields, which prevent coincidence events from being detected outside the axial plane of the emission. Retracting the septa increases the angle over which coincidence events are accepted and so improves the detector sensitivity and the count rate. However, the amount of detected scatter and random events also increases with wider acceptance angles, and a current debate focuses on whether retracted septa scanners can lead to improved reconstruction quality. Another drawback to retracting the septa is that the size of the reconstruction problem grows enormously, especially with algebraic approaches. In a 3D ML reconstruction with typical scanner geometries, the number of projections (rays of coincidence events) grows by an order of magnitude, and the size of the probability matrix, which is used throughout the ML reconstruction, can grow by four orders of magnitude or more. With the availability of high-performance parallel computer technology, we can now consider applying the ML algorithm to the problem of generating a full 3D PET reconstruction.

In addition to our program at NIH, there are many research groups throughout the United States that are applying parallel computing to biomedical problems (*3*). Much of this activity is part of the multiple agency, Federal High Performance Computing and Communications (HPCC) Program that has the goal of accelerating the development of future generations of high-performance computers and networks and the use of these resources throughout the American economy. For example, the National Science Foundation (NSF) activities have included the establishment of the following three Grand Challenge Application Groups: Advanced Computational Approaches to Biomolecular Modeling and Structure Determination, Understanding

Human Joint Mechanics Through Advanced Computational Models, and Imaging in Biological Research. In work jointly supported by NIH, NSF, and the Department of Energy, researchers at New York University, Sloan Kettering Cancer Center, and Oak Ridge National Laboratory are studying the effects of biochemically activated environmental chemicals on DNA.

## Exploiting Natural Concurrency

In developing computationally demanding biological applications, we divide a problem into many parts that can be independently executed on different processors. The speedup that can be achieved by having $n$ processors work concurrently on a computational problem is at most $n$ times faster than a single processor. Although attempts are made to achieve this ideal speedup, the ability to attain it in practice is determined by the efficiency of the developed parallel algorithm or method to exploit the natural concurrency in the computing problem. The actual speedup, which describes parallel performance, is defined as $S_n = T_1/T_n$, where $S_n$ is speedup on $n$ processors, $T_1$ is the time required to solve the problem on one processor, and $T_n$ is the time required for the parallel algorithm to solve the problem with $n$ processors. The efficiency of an algorithm is defined as the speedup per processor multiplied by 100, or $E_n = (S_n/n) \times 100$, with the ideal efficiency being 100%.

Three limitations on improving speedup are (i) the fraction of the computing task that is serial and can only execute on a single processor; (ii) the amount of parallelism in the task, that is, how much of the task can be divided into parts that can be independently executed on different processors; and (iii) the communication overhead, that is, the amount of time needed to transmit information between processors that cannot be overlapped with processing (*4*). If the serial fraction and the communication overhead times are small compared to the time required to complete the fraction of the task executed in parallel, the speedup approaches the ideal value for $n$ (*5*). A computational scientist makes this serial fraction small by choosing both a problem that is large enough to benefit from multiple processors and the proper parallel algorithm (*6*), which may or may not be the same one used for the same problem on a sequential system. The computational scientist also designs algorithms that minimize the amount of required interprocessor communication. The performance advantage attained with a parallel computer over a sequential computer depends on the type of problem solved and the way it is decomposed. Care is taken to reduce the overhead introduced by the parallel decomposition, because it can adversely affect the performance. We attempt to distribute the computing load evenly over the available processors throughout the computation. This last issue is what is referred to as the load balancing problem.

## Determining Virus Structure from Electron Micrographs

High-resolution cryoelectron microscopy in combination with 3D computer image reconstruction allows the structure of large icosahedral viruses to be studied. When the image noise is neglected and all the virus particles are assumed to have the same 3D structure, the specimens represented in the micrograph, which is a 2D projection along the line of sight, can be assumed to differ only in the orientations from which they are respectively viewed. These orientations must be determined before the reconstruction can be performed. A formalism that allows these orientations to be solved and a single 3D density map to be calculated from many noisy 2D projections was developed around 1970 (*7*). The introduction of cryoelectron microscopy in the early 1980s with its vastly improved image quality provided a stimulus for further refinement of the reconstruction procedures (*8, 9*). Further extension of the resolution, particularly for very large viruses such as the herpes simplex virus (*10*), will require the inclusion of much larger numbers of particles in the calculation, more finely sampled data, and refinement of the orientation angles to higher accuracy. Our approach to this challenge has been to mobilize the parallel computer, starting with the sequential methods provided by Baker and colleagues (*9*).

The 3D reconstruction of such viruses begins with one or more electron micrographs (Fig. 1), in which each particle on the micrograph is a 2D projection of a virus capsid specimen. The relative orientation of each specimen, denoted by polar and azimuthal angle pair $\{\theta,\phi\}$, defines the angle or view of the corresponding projection. If the effect of noise is neglected and all particles are assumed to be identical, the virus capsid specimens that generated the set of particles in the 2D micrograph can be considered identical in three dimensions except for their orientation. The reconstruction process takes advantage of the icosahedral symmetry of these viruses and the Fourier slice theorem (*7*), which states that the orientation of the particle plane in 3D Fourier space is identical to that of the projection plane. In Fourier space, each 2D projection is equivalent to 59 other symmetry-related views owing to the icosahedral symmetry, and these equivalent views all intersect at

**Table 1.** Applications of parallel computers to biomedical problems.

| Research area of biomedical interest | Representative scientific problem | Associated computationally intensive task |
|---|---|---|
| Structural biology: electron microscopy | Determine viral assembly mechanisms and identify individual protein components in capsid shells | Construct the 3D structure of viruses from electron micrographs |
| Structural biology: multidimensional NMR spectroscopy | Determine the 3D structure of proteins | Assign peaks in the NMR spectrum to atoms in the protein |
| Structural biology: x-ray crystallography | Determine the 3D structure of proteins | Perform crystallographic refinement for a protein molecule |
| Structural biology: protein chemistry | Predict the 3D structure of a protein from its primary structure | Calculate the solvent ASA of a protein |
| Computational chemistry: molecular dynamics | Study the kinetics of ultrafast chemical reactions in solution | Simulate chemical reactions using the technique of molecular dynamics |
| Computational chemistry: quantum mechanics | Determine the action of anti-cancer drugs on the DNA of cancer cells | Investigate the energetics of different cross-linked structures using ab initio quantum chemical methods |
| Biotechnology: genetic and protein sequence analysis | Discover relations and common motifs in nucleic acid and protein sequences | Analyze protein and nucleic acid sequences in large databases |
| Genetics: linkage analysis | Identify genes or regions of DNA involved in hereditary diseases | Calculate the log of the odds ratio needed to perform linkage analysis |
| Medical imaging: PET | Obtain PET Images of the brain that provide metabolic information | Reconstruct retracted-septa PET data using the expectation maximization algorithm |
| Medical imaging: radiation treatment planning | Find the optimal radiation treatment plan for a variety of human cancers | Calculate the 3D radiation dose distribution for a given patient anatomy and configuration of beams |

the Fourier origin. For each projected particle, the reconstruction algorithm interpolates that projection's Fourier transform, in a plane determined by its orientation parameters, onto a 3D grid in Fourier space.

Figure 2 shows the major steps in the reconstruction procedure starting with the cryoelectron microscopy and the digitization of the resulting micrograph with a microdensitometer. Preprocessing steps include selection of individual particles to be included in the reconstruction, removal of extraneous features such as an intruding neighbor particle or co-projected piece of contaminating ice, subtraction of the background gradient, normalization for constant mean and variance of the density values, and estimation of the center of each particle. Before we can perform a multiple particle reconstruction, we must obtain an estimate of each particle's orientation parameters, $\{\theta,\phi\}$. The steps involved in determining particle orientations are the most computationally intensive in the reconstruction process. We discuss the parallel implementation of the orientation estimation step, designated FindView, and the orientation refinement step, named Emicograd.

The FindView step uses the common lines procedure of Fourier analysis as implemented by Fuller (8) and Baker (9) to generate a set of candidate orientations for each particle. The common lines are pairs of radial lines in the Fourier transform of a particle projection along which the Fourier transform should be identical due to the icosahedral symmetry of the virus. Along any projection, there are 37 common line pairs that are specific to a particular orientation, $\{\theta_j,\phi_j\}$. Once correct orientations are selected from the FindView list of candidates, they are refined for each particle by Emicograd, which also uses common lines. Whereas FindView operates on each particle independently and separately, Emicograd uses information from all the particles in the set to refine the orientation parameters for each particular particle. This distinction leads to different parallel computing approaches: FindView replicates its data on each processor while Emicograd distributes its data among the processors.

FindView performs an exhaustive search through the orientation space to find those orientations whose common line Fourier transform values are most similar to those in the 2D Fourier transform of the particle projection (*11*). Icosahedral symmetry conditions permit the orientation search to be limited to an asymmetric search unit, which is defined to be a triangle with its apex at a threefold axis of symmetry and base vertices on fivefold axes of symmetry (Fig. 3, right). Orientations outside the search unit are mapped to an equivalent orientation inside this triangle. An exhaustive search scans the orientation space in approximately one degree or some other fixed increment over the entire asymmetric search unit. During the search, the required calculations at a given orientation are completely independent of the calculations at the other orientations. A processor can therefore perform the calculations over certain specified orientations while other processors complete the remaining orientations (Fig. 3, left). Every processor stores a replicate of the particle's projection image in memory. When the search is complete, the processors exchange a relatively small amount of data so that the best candidate orientations can be selected. The efficiency of the parallel implementation of FindView is almost the ideal 100% because it is easy to divide the search through the orientation space into independent tasks, the processors do not communicate until the end of the search, and the workload balance among the processors varies only slightly because of small differences in the number of independent common lines per orientation.

The orientation refinement step known as Emicograd refines the initial orientations supplied by FindView, producing a final set of particle projections to be used in the reconstruction. Performing calculations similar to those in FindView, Emicograd compares the values along the common lines of the Fourier transform of particle projections to find the particle orientation that minimizes the mean-squared error (cross-residual) between a given particle and all other particles. In a constrained refinement, the orientations of only one particle may be moved or refined while the others stay fixed. Constrained refinements are used to select the preferred orientation of a particular particle from the candidate orientations provided by FindView for all candi-

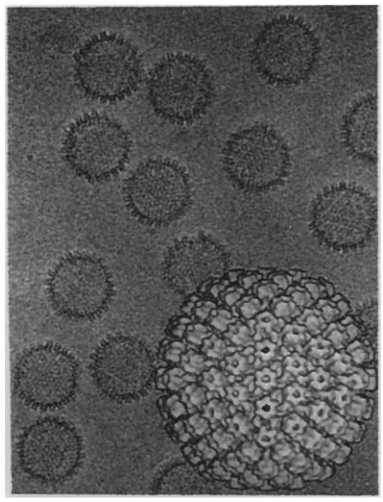

**Fig. 1.** (**Background**) A cryo-electron micrograph of herpes simplex virus (type 1) A capsids, empty of DNA. (**Foreground**) A surface-shaded representation of a 3D computer reconstruction obtained from 104 2D images (in collaboration with F. P. Booy, J. F. Conway, and A. C. Steven of NIH and W. W. Newcomb and J. C. Brown of the University of Virginia).

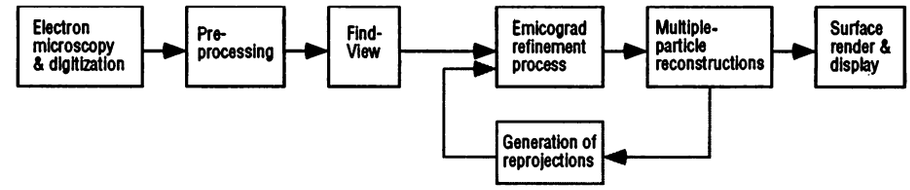


**Fig. 2.** Block diagram of steps involved in the 3D reconstruction of icosahedral virus images. The most computationally demanding steps are FindView, which generates initial estimates of individual particle orientations, and Emicograd, which refines particle orientations against others in the set.

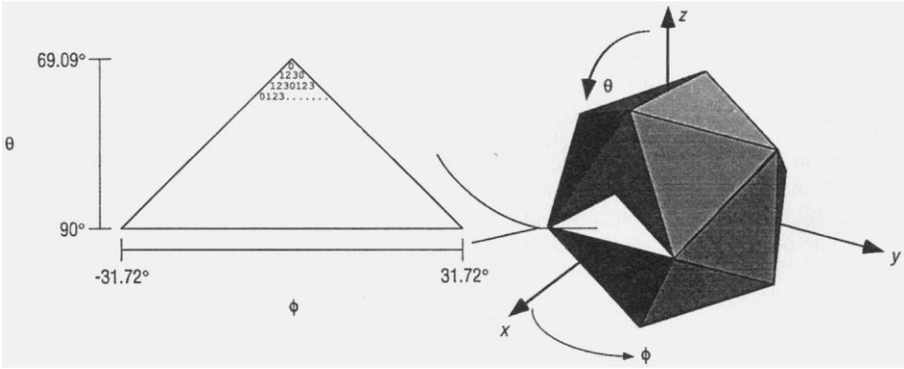


**Fig. 3.** (**Right**) Icosahedron showing orientation search parameters θ and ϕ. The asymmetric search unit is the white triangle. (**Left**) Asymmetric search unit illustrating the pattern of parallelization. Numbers inside the triangle represent processors assigned to that orientation in the search (four-processor case). With one-degree increments between angles, parallelization across 682 orientations provides good load balance up to at least 128 processors.

dates against a small, fixed, stable set. The large unconstrained refinements, which optimize the orientations of each particle against every other particle, are the most computationally demanding and are performed when the orientations of all images are optimized before a multiple particle reconstruction is performed. Memory and computational requirements of these large refinements have traditionally limited the number of particles allowed in the reconstruction.

The parallel implementation of Emicograd distributes the particle images across the processors and thereby substantially increases the number of particles that may be included in a reconstruction. This data distribution technique requires a significant amount of interprocessor communication in the computation of the refinement criteria. Consequently, the parallel performance of Emicograd is less favorable than that of FindView. A DEC (Digital Equipment Corporation, Maynard, Massachusetts) VAX 4000-500 processor requires 10.4 min to run the FindView search on a single particle while a single processor of an Intel (Intel Supercomputer Systems Division, Beaverton, Oregon) iPSC/860 parallel computer requires 7.1 min and 64 iPSC/860 processors together require 7.2 s, a speedup of 59. In contrast, an unconstrained refinement of 55 particles on a VAX 4000-500 takes 110 min to complete. This same process runs in 4.3 min on 64 iPSC/860 processors, a speedup of only 38 from the single processor time of 165 min. To put these times in perspective, in a recent reconstruction involving 335 particles and 32 iPSC/860 processors, 5.5 total compute hours were spent on FindView while 33.5 total compute hours were spent on constrained and unconstrained Emicograd refinements.

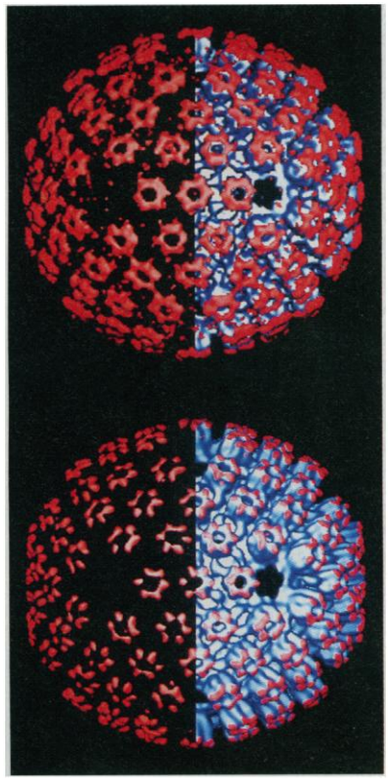

**Fig. 4.** Shown in red is the location of a minor capsid protein called VP26 (12 kD) (*12*) from herpes simplex virus (type 1). This protein was mapped in experiments in which it was first extracted from purified capsids by treatment with guanidine hydrochloride (*13*), purified, and then rebound to the capsids. The top right image shows the depleted capsid (blue) and the additional density (red) visualized when VP26 was rebound. The two 3D reconstructions in the difference map (top left image) are based on 35 and 40 images, respectively, which could have been processed with a conventional sequential computer. Parallel computing extended the analysis to include 135 and 167 images, respectively, yielding the bottom image. The resolution was thus improved from approximately 3.5 to under 3.0 nm, and the signal-to-noise ratio was substantially improved. The clusters of six VP26 subunits, shown blurred together in the top image, are clearly resolved in the bottom image (in collaboration with F. P. Booy, J. F. Conway, and A. C. Steven of NIH and W. W. Newcomb and J. C. Brown of the University of Virginia).

As shown in Fig. 4, Booy and colleagues (*12*) recently reported locating a small protein, VP26 (*13*), on the hexomer tips of the human herpes simplex virus capsid. This detection of six distinct subunits was, in part, due to the signal-to-noise improvements that have been achieved as a result of the larger number of particles in the reconstruction. The figure demonstrates the difference between earlier reconstructions with 35 and 40 particles and more recent reconstructions with 135 and 167 particles.

## Protein Structure Prediction

Since Anfinsen (*14*) first noted that the information needed to determine how a protein folds resides completely within its amino acid sequence, the problem of predicting protein folding has been one of the most important unsolved problems in biochemistry. Understanding the 3D structure of proteins is important to studying their function in living systems and designing new ones for biological and medical purposes. The amino acid sequences of proteins are being discovered at an explosive rate. However, experimental procedures for determining their 3D structure, such as x-ray crystallography and NMR spectroscopy, are slow, costly, and complex. A need exists for theoretical and computational techniques that can be used to help predict the structure from the sequence.

The protein folding problem remains unsolved because all of the biochemical rules that govern the folding and stability of proteins are not yet known. If these rules were known, a computer program could be written to simulate the folding of a protein. In conjunction with the scientific work that is being done to understand the forces involved in protein folding, an alternative computer approach is to write a program that searches through all the possible protein conformations to find the ideal one. However, a search through the entire conformation space would require a prohibitive amount of computer time. The popular lattice-space Monte Carlo method (*15*) has been used to reduce the number of possible protein conformations significantly by confining a protein onto a lattice. This method does not always yield a useful result because of the distortion introduced in the predicted structure by reducing the possible number of conformations.

At NIH, we are developing parallel computing methods for simulating protein folding so that more possible conformations can be considered and a more realistic energy function can be computed. This work involves strategies for searching through a large number of possible structures representing different energy states. The computationally intensive parts of a simulation are the long search through the great number of possible conformations and the computation of the free energy of the structures being considered during the simulation. We use a potential energy function that is the sum of three energy terms (*16*, *17*). The first term is an effective potential based on the $\phi,\psi$ angle probabilities of the protein's main chain. The second term is due to hydrogen bonding between the residues of the protein. The time to calculate these two terms is short. The third term is the hydrophobic potential that is proportional to the solvent-accessible surface area (ASA) of the protein molecule. The calculation of the ASA of the numerous structures considered in the simulation requires parallel computer performance.

We have used the Lee and Richards algorithm (*18*) to determine the ASA of a protein. With this algorithm, we consider a protein molecule as a set of interlocking spheres, one for each atom of the protein. The radius of the sphere for an atom, $j$, is given by the sum of the van der Waals radii of the atom $j$ and the chosen solvent. In our work, the Lee and Richards set of the van der Waals radii are used and the solvent radius $R_s$ was set to 1.4 Å, corresponding to that of water.

The entire set of interlocking spheres is sectioned by a set of parallel planes perpendicular to the $z$ axis with a predetermined spacing, $\Delta Z$. The intersection of each

sphere with a given plane appears as a circle. The arcs of a circle that are within other circles are eliminated. The drawing in any one plane thus becomes the trace of the accessible surface on the given section (Fig. 5). If any part of the circle for a given atom is drawn, the atom is accessible and the summed length of all arcs for the atom will be a measure of the ASA of the atom in that plane. The total ASA of the atom is given by numerical integration of all the summed arc lengths appearing in all sections (*19*).

An evaluation of the equation for determining the ASA of a protein reveals that the ASA calculation can be performed on individual atoms separately, and independently, if the location of the neighboring atoms of each atom is known. We can use a data replication method in which each processor keeps a complete list of atomic coordinates of all the atoms in the protein structure. Subsequently, no communication is required between processors until the end of the computation. On the basis of these observations, we implemented the following parallel algorithm (*20*): (i) Read the atomic coordinates and broadcast all $N$ atomic coordinates to each processor, allowing each processor to keep a complete list of the atomic coordinates. (ii) Partition the atoms, which represent the computing workload, among the $n$ processors: For each processor $i$, where $1 \leq i \leq n$, determine a set of atoms for which it must compute the ASA; these atoms are defined in the range (*start*[$i$], *end*[$i$]) for processor $i$, where $1 \leq$ *start*[$i$],*end*[$i$] $\leq N$. (iii) In parallel, for all processors $i$: First, find the neighboring atoms, $\eta j$, for each atom $j$ that is assigned to processor $i$, that is, for *start*[$i$] $\leq j \leq$ *end*[$i$]; next, once the neighboring atoms are identified, compute the ASA of its assigned atoms. (iv) Compute the global sum of the ASAs computed by each processor. The synchronization and communication between processors are arranged to occur at this step to complete this global summation.

Partitioning and mapping the atoms across the $n$ processors in step (ii) is done with the objective of balancing the workload. An obvious scheme is a uniform partitioning and mapping of the atoms across the processors; that is, allocate an equal number of $N/n$ atoms to each processor such that processor $i$ gets *start*[$i$] = $(i - 1)N/n + 1$ to *end*[$i$] = $iN/n$. In a dynamic protein folding simulation, the ASA algorithm is invoked to compute the ASA for each of the millions of sequentially formed protein structures. In the sequence of structures, the atomic coordinates of the atoms change because of the movement of the atoms in the protein structure. Therefore, the number of neighbor atoms for each atom changes from structure to structure, and the workload varies accordingly. This change is especially severe in protein folding studies in which the structure of the protein changes from a loosely coiled string to a compact globular form. A mapping that was optimal at the start of the folding process may be inefficient after a number of folding iterations. Therefore, it is necessary to remap the workload dynamically among the processors, that is, to repartition the atoms as the simulation progresses.

A remapping scheme must (i) determine when a remapping is necessary, (ii) dynamically estimate and compute the workload distribution, and (iii) compute an efficient mapping of the workload to the processors. To reduce overhead, we perform a remapping after a specified number of iterations in the simulation. We estimate the workload of each atom $j$ by its number of neighboring atoms, $\eta j$. The total computational workload for the entire protein molecule is the sum of the individual atom workloads, $\Sigma_j \eta_j$, and the average workload is this sum divided by the total number of processors, $n$. We then map the atoms to the processors so that each processor is assigned atoms to the point where its assigned workload is equal to or just exceeds the average workload.

The parallel implementation of the ASA calculation has allowed us to include this factor in our potential energy function. A Silicon Graphics (Mountain View, California) Indigo R3000 processor required 48.5 s to perform a single ASA calculation for a 333-residue protein. The same ASA calculation was completed in 0.8 s on 64 processors of an Intel iPSC/860 parallel computer. This difference approximately reduces every hour of computer time needed to calculate the ASA during a protein folding simulation on a workstation to 1 min on the parallel computer with 64 processors.

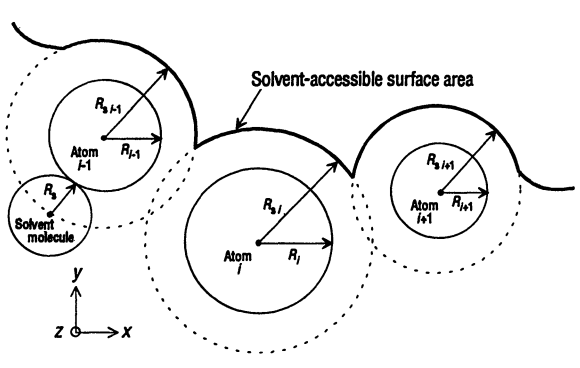


**Fig. 5.** When the solvent ASA is determined, the protein is sectioned by a set of parallel planes perpendicular to the $z$ axis with fixed spacing between planes. The radius for an atom is given by the sum of the van der Waals radii of that atom and the chosen solvent. The trace of the accessible surface can be seen where the arcs of the atoms do not overlap.

## Genetic Database Searching

The human genome project has generated a massive volume of genetic data, such as that found in the GenBank (*21*). This database (release 82.0 from the National Center for Biotechnology Information at the National Library of Medicine of NIH) contains 169,896 DNA sequences with 180,589,455 bases and their related biological and bibliographical information. Currently, GenBank not only contains human DNA sequences but also the sequences of plants, viruses, bacteria, and other species.

One of the standard methods used to acquire new information from GenBank is to search the entire database for homologous sequences. We search for patterns across different species as well as within the same one. When we discover a new sequence, we search the database for sequences that are similar or relevant to this discovery. In addition, we often search the database at regular intervals because new sequences are added daily. As the size of GenBank increases, it remains critical to keep the search time short.

To search for the most homologous sequence to a given query sequence, we need an algorithm that defines the similarity between two sequences. That is, we must first determine the degree of similarity between two sequences. The preferred comparison algorithm was developed by Smith and Waterman (*22*) and later modified by Gotoh (*23*). This algorithm uses a dynamic programming technique to compare two sequences (*24*) and requires $M \times N$ computational steps to find a similarity score between two sequences of respective lengths $M$ and $N$.

We have developed a parallel sequence searching method (*25*) in which the original sequences in the database are placed into one of $n$ buckets (smaller databases), where $n$ is the largest number of processors, so that the difference between the sum of the sequence lengths in the smallest and those in the largest buckets is minimized. In other words, each bucket must have approximately the same number of bases. When the original sequences are divided this way, each processor can search its own bucket without communicating with other processors if all $n$ processors are used. If only $n/2$ processors are used, each processor can search two buckets. The number of buckets

is always a multiple of the number of processors. Once all the processors find their best local homologous sequence, the best global homologous sequence can also be found. A number of other groups have developed parallel sequence searching methods (26), and we compare our method with theirs in reference (25).

We implemented an algorithm that places the sequences of the original database into one of the $n$ buckets. First, the sequences must be sorted in decreasing length order. Then, starting from the longest one, each sequence is placed into the bucket that has the smallest sum of sequence lengths until all the sequences in the database are assigned to a bucket. The distribution algorithm required less than 30 s to place the 169,896 sequences of GenBank into 128 buckets on a desktop workstation. We can determine the workload balance, that is, the difference between the largest amount of sequence comparison work any processor will have and the least amount of sequence comparison work any processor will have, for this distribution of sequences among the processors by computing the percentage of workload imbalance, given by

$$\left(\frac{\text{Largest bucket} - \text{Smallest bucket}}{\text{Largest bucket}}\right) \times 100$$

Table 2 shows that for our sequence distribution algorithm the imbalance is very small for the entire range of processor numbers.

To minimize the communication overhead, we fetch only one sequence at a time instead of loading the entire database into memory at one time. The next sequence is fetched while a processor compares the previously fetched one. The fetch time can then be overlapped with the processing time. There is incomplete overlap for only the first sequence and occasionally after a short sequence has been fetched. To expedite the sequence fetching process, we store the sequences so that an unformatted read function, which is faster than the formatted one, can be used to fetch a sequence and only one read operation is needed to fetch each sequence entry.

Our parallel sequence searching method achieved ideal speedup when implemented on an Intel iPSC/860 parallel computer. A single iPSC/860 processor required approximately the same amount of time to complete a GenBank query as a SUN SPARCstation 2 processor, so the multiple of improvement over this type of workstation is simply the number of processors used on the parallel system. For example, a GenBank query for the human gene for tyrosine aminotransferase with 1600 base pairs required approximately 120 hours on a single processor but finished in less than 2 hours with 64 processors of the iPSC/860 parallel computer.

**Table 2.** The percentage of workload imbalance when the sequences of the GenBank database are divided into buckets.

| Number of processors | Percentage of load imbalance |
|---|---|
| 2 | 0.00000 |
| 4 | 0.00000 |
| 8 | 0.00002 |
| 16 | 0.00005 |
| 32 | 0.00006 |
| 64 | 0.00033 |
| 128 | 0.00033 |

## Future Prospects

Parallel computing can be used effectively in a variety of biomedical problems. The high computational efficiencies we obtained for the three illustrated applications are shown in Table 3 (27). For the 3D reconstruction of virus capsids from electron micrographs, the increased performance attained with parallel computing allows us to use more virus particles in their reconstructions. Increasing the number of particles has significantly improved the signal-to-noise ratios of the 3D reconstructions. In the future, we hope to obtain improved resolution of the reconstructions by a combination of new methods. Parallel computing can greatly reduce the time required to calculate the solvent ASA of a protein molecule. This reduction makes it possible to use this important measure in dynamical simulations of proteins where a million or more structures must be considered. Finally, as the number of DNA and amino acid sequences in GenBank and other databases grows exponentially, we will need the computational tools to analyze the sequence patterns contained therein. As can be seen from our implementation of a parallel sequence analysis method, parallel computing can provide the performance we need to use large databases in our research.

**Table 3.** Efficiencies for the parallel implementation of three computationally intensive biomedical applications: A, finding the initial orientation of a single herpes simplex virus particle with FindView; B, calculating the solvent ASA of a 2556-atom protein; and C, comparing a 1335-base DNA sequence with the GenBank database (release 82.0).

| Number of processors, $n$ | Application efficiency, $S_n$ | | |
|---|---|---|---|
| | A | B | C |
| 1 | 100 | 100 | 100 |
| 2 | 100 | 98 | 100 |
| 4 | 99 | 97 | 100 |
| 8 | 99 | 93 | 100 |
| 16 | 98 | 88 | 100 |
| 32 | 96 | 82 | 100 |
| 64 | 91 | 67 | 99 |

## REFERENCES AND NOTES


1. F. Delaglio, private communication.
2. K. Lange and R. Carson, *J. Comp. Assist. Tomogr.* **8**, 306 (1984).
3. *High Performance Computing and Communications: Toward a National Information Infrastructure*, Report by the U.S. Committee of Physical, Mathematical, and Engineering Sciences of the Federal Coordinating Council for Science, Engineering and Technology; and the Office of Science and Technology Policy (CPMES of FCCSET and OSTP, National Science Foundation, Washington, DC, 1994); T. C. Pilkington *et al.*, *High-Performance Computing in Biomedical Research* (CRC Press, Boca Raton, FL, 1993).
4. G. S. Almasi and A. Gottlieb, *Highly Parallel Computing* (Benjamin/Cummings, Redwood City, CA, 1994).
5. A simple expression for $T_n$, approximating the relation of $T_n$ to $T_1$, demonstrates the effect of these three factors:

$$T_n = f \times T_1 + \frac{(1T - f)T_1}{n} + \frac{T_1}{r}$$

where $f$ is the fraction of the work that must be done by a single processor. The communication time is represented by the $T_1/r$ term, where $r$ is the ratio of the time to solve the problem on one node to the communication time required with $n$ processors. Substituting this expression into the one for speedup leads to the following:

$$S_n = \frac{1}{f + (1 - f)/n + (1/r)}$$

As the serial fraction approaches zero and the communication fraction becomes large, the speedup approaches the ideal $n$.
6. W. D. Hillis and B. M. Boghosian, *Science* **261**, 856 (1993); F. Baskett and J. L. Hennessy, *ibid.*, p. 864.
7. R. A. Crowther, *Philos. Trans. R. Soc. London Ser. B* **261**, 221 (1971).
8. S. D. Fuller, *Cell* **48**, 923 (1987).
9. T. S. Baker, J. Drak, M. Bina, *Proc. Natl. Acad. Sci. U.S.A.* **85**, 422 (1988).
10. J. F. Conway *et al.*, *J. Struct. Biol.* **111**, 222 (1993).
11. The set of common line pairs, denoted by angles $\alpha\ \varphi_1(k)$ and $\alpha\ \varphi_2(k)$, where $k = 1 \ldots 37$, are specific to a particular orientation $\{\theta_j,\phi_j\}$. An exhaustive search through orientation space might find those orientations whose common line Fourier transform values are most similar in a mean-squared sense. FindView reads an image $f(x,y)$ corresponding to a single projected particle view; takes a discrete Fourier transform, $F(u, v) = \Im\{f(x,y)\}$; and interpolates the transform onto a polar grid $F(u,v) \xrightarrow{\text{polar}} F(\rho/\omega)$. In a noise-free environment along the common lines, if the orientation is known the Fourier transforms (and hence the phase of the Fourier transform) should be equal. Defining $F(\rho,\omega) = |F(\rho,\omega)| \times \exp[i \times \psi(\rho,\omega)]$, we seek the orientation $\{\theta_j,\phi_j\}$ that minimizes $\delta\psi[\rho,k;\varphi] = \psi(\rho,\alpha\ \varphi_2[k]) - \psi(\rho,\alpha\ \varphi_2[k])$ for all common line pairs $k = 1 \ldots 37$ and over the range of band-pass Fourier radii $\rho_{min} \ldots \rho_{max}$. For each unique orientation we compute an average phase residual (mean squared error)

$$R[j] = \frac{\sqrt{\sum_{k=1}^{37} \sum_{\rho=\rho_{min}}^{\rho_{max}} (\delta\psi[\rho, k; j])^2}}{37\ (\rho_{max} - \rho_{min})}$$

Those orientations with the lowest phase residuals are saved as the best candidate orientations for the particle.

The phase residual equation is true only if all 37 common line pairs are independent. In general, 37 must be replaced with $37 - J(\theta_j,\phi_j)$, the number of independent common-line pairs. A third orienta-

tion parameter, $\Omega$, represents the angular position of the common lines relative to the Cartesian axes. In addition to searching for the $\{\theta,\phi\}$, FindView rotates the $\underline{\alpha}$ pairs of a given orientation by $\underline{\alpha} + \Omega,\Omega = 1°, 2°, \ldots, 180°$ to determine the rotation angle $\Omega_m$ that provides the lowest phase residual $R[m;j]$. Recognizing that the residual matrix favors those orientations with the fewest independent common-line pairs, we use a $\chi^2$ weighting scheme that uses $37 - J(\theta,\phi)$ degrees of freedom per orientation. The residual is weighted according to the probability that a $\chi^2$ variable will exceed the average residual $R[m;j]$ at the chosen $\Omega_m$, normalized with respect to the average $\overline{R}_j$ over all $\Omega$.

12. F. P. Booy *et al.*, *Proc. Natl. Acad. Sci. U.S.A.* **91**, 5652 (1994).
13. W. W. Newcomb *et al.*, *J. Mol. Biol.* **232**, 499 (1993).
14. C. B. Anfinsen, *Science* **181**, 223 (1973).
15. H. S. Chan and K. A. Dill, *Phys. Today* **46**, 24 (February 1993); D. G. Covell and R. L. Jernigan, *Biochemistry* **29**, 3287 (1990); D. A. Hinds and M. Levitt, *Proc. Natl. Acad. Sci. U.S.A.* **89**, 2536 (1992).
16. H. S. Kang, N. A. Kurochkina, B. Lee, *J. Mol. Biol.* **229**, 448 (1993).
17. B. Lee, private communication.
18. _____ and F. M. Richards, *J. Mol. Biol.* **55**, 379 (1971).
19. The accessible surface area $A_j$ of atom $j$ is given by the following equation:

$$A_jT = TR_{sj}\sum_k \frac{D_{jk} \times L_{jk}}{\sqrt{R_{sj}^2 - Z_{jk}^2}}$$

where the summation is over all sections $k$, $R_{sj}$ is the sum of the van der Waal's radii of the solvent and the atom $j$, $L_{jk}$ is the summed length of the arcs drawn on section $k$, $Z_{jk}$ is the perpendicular distance from the center of the sphere to the section $k$, and $D_{jk} = \Delta Z/(2 + \Delta Z_{jk})$, where $\Delta' Z_{jk} = \min(\Delta Z/2, R_{sj} - Z_{jk})$.

20. E. B. Suh, B. Lee, B. Narahari, A. Choudhary, R. L. Martino, *Proc. Seventh Int. Parallel Process. Symp.* (1993), p. 685.
21. C. Burks *et al.*, *Nucleic Acids Res.* **19** (suppl.), 2221 (1991).
22. T. F. Smith and M. S. Waterman, *J. Mol. Biol.* **147**, 195 (1981).
23. O. Gotoh, *ibid.* **162**, 705 (1982).
24. Let the two sequences be $A = a_1a_2a_3 \ldots a_M$ and $B = b_1b_2b_3 \ldots b_N$ and let sub$(a_1,b_j)$ be a given similarity score for substituting residue $a_i$ by $b_j$. The penalty for introducing a gap into a sequence is defined by the gap penalty function $w_k = uk - v$, $(u, v \geq 0)$ where $k$ is the gap length. The Gotoh algorithm is given as follows:

$$S_{i,j} = \text{Max}\begin{cases} P_{i,j} \\ S_{i-1,j-1} + \text{sub}(a_i,b_j) \\ Q_{i,j} \end{cases}$$

$$P_{i,j} = \text{Max}[S_{i-1,j} + w_1, P_{i-1,j} + u]$$

$$Q_{i,j} = \text{Max}[S_{i,j-1} + w_1, Q_{i,j-1} + u]$$

where $S_{i,j}$ is the accumulative similarity score between sequences A and B up to the $i$th and $j$th positions. The initial conditions can be defined as follows: $S_{i,0} = P_{i,0} = Q_{i,0} = 0$, and $S_{0,j} = P_{0,j} = Q_{0,j} = 0$ for $0 \leq i \leq M$ and $0 \leq j \leq N$.

25. T. K. Yap, O. Frieder, R. L. Martino, in *Handbook of Parallel and Distributed Computing*, A. Y. Zomaya, Ed. (McGraw-Hill, New York, in press).
26. X. Guan, R. Mural, R. Mann, E. Uberbacher, *Proc. of the Fifth SIAM Conf. Parallel Process Sci. Comput.* (1991), p. 332; R. Jones, *Comput. Appl. Biosci.* **8**, 377 (1992); D. F. Sittig, D. Foulser, N. Carriero, G. McCorkle, P. L. Miller, *Comput. Biomed. Res.* **24**, 152 (1991).
27. We implemented the three biological applications that we described in detail on the NIH Intel Supercomputer Systems Division iPSC/860 parallel computer. It has 128 computer nodes with 16 megabytes of memory per node. A high-speed hypercube network connects all the computer nodes of the system. The Intel Concurrent I/O File System, consisting of 15 disks and eight I/O nodes that communicate with the processor nodes through the hypercube interconnect, provides a 10-gigabyte, fast access, mass storage facility.
28. We thank the following NIH people for important contributions to various aspects of this parallel computing effort: J. C. Pfeifer and N. I. Weisenfeld of the Computational Bioscience and Engineering Laboratory, Division of Computer Research and Technology (DCRT), NIH; D. Rodbard, the Director of DCRT; A. C. Steven, F. P. Booy, and J. F. Conway of the Laboratory of Structural Biology, National Institute of Arthritis, Musculoskeletal and Skin Diseases; B. Lee of the Laboratory of Molecular Biology, National Cancer Institute; and F. Delaglio of the Laboratory of Chemical Biology, National Institute of Diabetes and Digestive and Kidney Diseases. We also thank the following for their contributions: S. Erwin of Intel Supercomputer Systems, W. W. Newcomb and J. C. Brown of the University of Virginia, T. S. Baker of Purdue University, B. Narahari of George Washington University, and O. Frieder of George Mason University.