CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2005; (in press)
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe.950

Parallel protein folding with STAPL

Shawna Thomas[1], Gabriel Tanase[1], Lucia K. Dale[2], Jose M. Moreira[3], Lawrence Rauchwerger[1] and Nancy M. Amato[1,*,†]

[1]*Parasol Laboratory, Department of Computer Science, Texas A&M University, College Station, TX 77843-3112, U.S.A.*
[2]*University of the South, Sewanee, TN 37383-1000, U.S.A.*
[3]*IBM TJ Watson Research Center, Yorktown Heights, NY 10598-0218, U.S.A.*

SUMMARY

The protein-folding problem is a study of how a protein dynamically folds to its so-called native state—an energetically stable, three-dimensional conformation. Understanding this process is of great practical importance since some devastating diseases such as Alzheimer's and bovine spongiform encephalopathy (Mad Cow) are associated with the misfolding of proteins. We have developed a new computational technique for studying protein folding that is based on probabilistic roadmap methods for motion planning. Our technique yields an approximate map of a protein's potential energy landscape that contains thousands of feasible folding pathways. We have validated our method against known experimental results. Other simulation techniques, such as molecular dynamics or Monte Carlo methods, require many orders of magnitude more time to produce a single, partial trajectory. In this paper we report on our experiences parallelizing our method using STAPL (Standard Template Adaptive Parallel Library) that is being developed in the Parasol Lab at Texas A&M. An efficient parallel version will enable us to study larger proteins with increased accuracy. We demonstrate how STAPL enables portable efficiency across multiple platforms, ranging from small Linux clusters to massively parallel machines such as IBM's BlueGene/L, without user code modification. Copyright © 2005 John Wiley & Sons, Ltd.

KEY WORDS: protein folding; motion planning; parallel libraries; C++

*Correspondence to: Nancy M. Amato, Parasol Laboratory, Department of Computer Science, Texas A&M University, College Station, TX 77843-3112, U.S.A.
†E-mail: amato@cs.tamu.edu

Contract/grant sponsor: NSF; contract/grant numbers: ACI-9872126, EIA-9975018, EIA-0103742, EIA-9805823, ACR-0081510, ACR-0113971, CCR-0113974, EIA-9810937, EIA-0079874 and IIS-0118097
Contract/grant sponsor: DOE; contract/grant number: B520713
Contract/grant sponsor: Texas Higher Education Coordinating Board; contract/grant number: ATP-000512-0261-2001

Received December 2003
Revised May 2004
Accepted September 2004

INTRODUCTION

Protein folding research is typically focused on two main problems: protein structure prediction and the study of the protein folding process. Determining a protein's native three-dimensional structure is important because the protein's function is related to it [1,2]. Knowledge of the folding process is of great practical importance since some devastating diseases such as Alzheimer's and bovine spongiform encephalopathy (Mad Cow) are associated with misfolded proteins [3]. In our research, we assume the native structure is known and concentrate on studying the protein-folding mechanism. We study issues related to the folding process such as secondary and tertiary structure formation and its dependence on the initial denatured configuration.

In previous work [4–6], we developed a new computational technique for studying protein folding. It is based on a class of motion planning techniques called probabilistic roadmap methods (PRMs) [7] that have proven effective on many diverse applications ranging from robotics, to computer animation, to molecules. Our technique yields an approximate map of a protein's potential- and free-energy landscapes that contains thousands of feasible folding pathways and enables the study of global properties of the folding landscape. We obtained promising results for several small proteins (60–100 amino acids) [4] and validated our pathways by comparing secondary structure formation order with known experimental results [8]. In one case study, we demonstrated that our technique is sensitive enough to identify subtle differences in folding behaviors for structurally similar Proteins G and L [6].

Our technique, although significantly more computationally efficient than previous methods, still requires large amounts of computational resources. For example, it takes several hours on a desktop PC to compute a map approximating the potential landscape of a small (60 residue) protein when using a coarse approximation for the energy calculations. With a more accurate and detailed energy calculation, the running time increases to two weeks. It is imperative that we find a faster technique if we are to study larger proteins with higher accuracy. Fortunately, PRMs are 'embarrassingly parallel' [9]. In particular, as we will see, the running time of our technique is dominated by energy calculations, and most are independent of each other.

In this paper, we describe how we used the Standard Template Adaptive Parallel Library (STAPL) [10–12] to parallelize our existing PRM-based protein folding code. We chose STAPL for the following reasons: (i) STAPL allows for an easy transition from sequential code to parallel code by extending the ANSI C++ Standard Template Library (STL) [13]; and (ii) STAPL provides portable efficiency to different systems, both shared-memory and distributed-memory models, without requiring user code modification. We present experimental results showing good speedups on multiple platforms, ranging from small Linux clusters, to distributed shared-memory machines such as SGI's Altix, to massively parallel machines such as IBM's BlueGene/L. We demonstrate how STAPL enables one to run the same parallel application on several different platforms without modifying user code.

RELATED WORK

Protein folding

Several computational approaches have been applied to the protein folding problem, see Table I. These include lattice models [14], energy minimization [15,16], molecular dynamics [17–20], Monte Carlo methods [21,22], and genetic algorithms [23,24]. Molecular dynamics and Monte Carlo methods

Table I. A comparison of protein-folding models.

Approach	Folding landscape	Number of paths produced	Path quality	Compute time	Native state required
Molecular dynamics	No	1	Good	Long	No
Monte Carlo	No	1	Good	Long	No
Statistical model	Yes	0	N/A	Fast	Yes
PRM-based	Yes	Many	Approximate	Fast	Yes
Lattice model	Not used on real proteins				

provide a single, high quality folding pathway, but each run is computationally intensive. Statistical mechanical models [25–27], while computationally more efficient, are limited to studying global averages of folding kinetics and are unable to produce folding pathways. Our PRM-based work provides an alternative approach that computes a map approximating the energy landscape which contains thousands of approximate folding pathways for the given protein.

Subsequent to our initial work, Apaydin *et al*. [28,29] have also used PRM-based techniques to study protein folding. Their work differs from ours in some important aspects. First, they model the protein at a much coarser level considering each secondary structure to be already formed (i.e. rigid). Second, while our approach has been to validate our results by comparing with experimental data, they have compared the PRM approach with computational techniques such as Monte Carlo simulation.

PRM

Given a description of the environment and a movable object, the motion planning problem is to find a valid path taking the movable object from a start configuration to a goal configuration [30]. The PRM [7] has been shown to be highly successful in solving the motion planning problem for objects with many degrees of freedom (DOF).

PRMs work by first sampling random points in the movable object's configuration space (C-space), which is the set of all possible positions and orientations of the movable object, valid or not [31]. Only those samples that meet certain feasibility requirements (e.g. collision free or potential energy less than some threshold) are kept. The samples are connected to form a graph (or roadmap) by using some simple local planner (e.g. a straight line in C-space) to connect nearby points. The roadmap can then be used to answer queries corresponding to different start and goal pairs. To answer a query, the start and goal are connected to the roadmap. Then a path from the start to the goal is extracted from the roadmap if it exists. A major strength of PRMs is that they are simple to apply, even for high DOF problems, only requiring the ability to randomly sample points in C-space and test them for feasibility.

Parallel techniques

Parallel protein-folding techniques have mainly been restricted to molecular dynamics simulations [32,33]. Weiner and Kollman pioneered work in this area with AMBER [34]. NAMD [35,36] is a parallel molecular dynamics code designed for high-end parallel machines. It has been shown to scale to hundreds of processors on massively parallel machines and to tens of processors on PC clusters. Folding@Home [37–39] uses a distributed computing technique to run molecular dynamics simulations. It overcomes the huge computation barrier of molecular dynamics simulations by making use of over 40 000 machines. Still, only relatively small motions have been simulated with this method [40].

There has also been some work on parallel motion planning. Lozano-Pérez and O'Donnell [41] developed a parallel algorithm for computing a discretized C-space for the first three links of a six DOF articulated linkage. Challou *et al.* [42,43] parallelized the Randomized Path Planner [44], a randomized potential field method. Finally, in [9] it was shown that PRMs are 'embarrassingly parallel' and impressive speedups were given.

PROTEIN FOLDING USING PRMS

In previous work, we successfully applied the PRM framework to study protein-folding pathways [4–6]. The protein is modeled as an articulated linkage. Using a standard modeling assumption for proteins that bond angles and bond lengths are fixed [45], the DOF in our model are the backbone's phi and psi torsional angles. These are modeled as revolute joints taking values in the range $[0, 2\pi)$.

PRMs can be applied to proteins by simply replacing the traditional collision-free requirement with a potential energy calculation. A sample q is accepted with the following probability where $E(q)$ is the potential energy:

$$P(\text{accept } q) = \begin{cases} 1 & \text{if } E(q) < E_{\text{min}} \\ \dfrac{E_{\text{max}} - E(q)}{E_{\text{max}} - E_{\text{min}}} & \text{if } E_{\text{min}} \leq E(q) \leq E_{\text{max}} \\ 0 & \text{if } E(q) > E_{\text{max}} \end{cases}$$

Due to the high dimensionality of the protein's C-space, uniform sampling would take too long to provide sufficiently dense coverage of the region surrounding the native state. Instead, we bias our sampling around the native state by iteratively applying small perturbations to existing configurations. Node connection is done in the same way as traditional PRMs except that each connection is assigned a weight to reflect its energetic feasibility. This allows us to easily search for low-energy paths in the roadmap.

We obtained promising results for several small proteins (60–80 amino acids) and validated our pathways by comparing the secondary structure formation order with known experimental results [8] using timed contact analysis. We also demonstrated that this technique is sensitive enough to identify the different folding behaviors of structurally similar Proteins G and L [6].

Roadmap analysis

Timed contact analysis gives us a formal method of validation and allows for detailed analysis of the folding pathways. We first identify the *native contacts* by finding all pairs of C_α atoms in the native state that are at most 7 Å apart. If desired, attention can be restricted to *hydrophobic contacts* between hydrophobic residues. To analyze a pathway, we examine each conformation on the path and determine

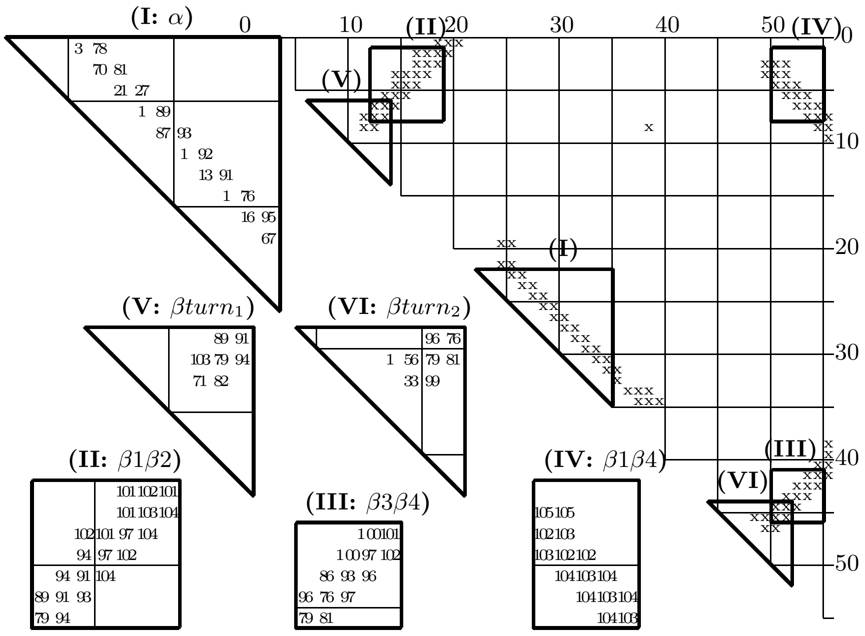

Figure 1. Timed contact map for Protein G. The full contact matrix (right) and blow-ups (left) showing the time steps when contacts appear on a path. The blow-ups: I: α helix, II: $\beta1\beta2$, III: $\beta3\beta4$, IV: $\beta1\beta4$, V: turn 1 ($\beta1\beta2$), and VI: turn 2 ($\beta3\beta4$).

the time step at which each native contact appears. Although these time steps cannot be associated with any real time, they give a temporal ordering and produce a *timed contact map* for the pathway. Figure 1 shows a timed contact map for Protein G.

The timed contact map provides a formal basis for determining secondary structure formation order along a single pathway. Here, structure formation order is based on the formation order of the native contacts [46]. We have looked at several metrics to determine when a secondary structure appears: average appearance time of native contacts within the structure, average appearance of the first $x\%$ of the contacts, average appearance ignoring outliers, etc. We can also focus our analysis on smaller pieces of secondary structure such as β-turns (instead of the entire β-sheet). This is especially helpful when looking for fine details in a pathway.

We can also use the roadmap to study more general properties of the protein's folding behavior. For example, if the roadmap maps the potential energy landscape well, then the percentage of pathways in the roadmap that contain a particular formation order should reflect the probability of that order occurring. Table II shows secondary structure formation order results for Proteins G and L [6]. Our results were able to identify the different folding behaviors of the two structurally similar proteins.

Table II. Comparison of analysis techniques for Proteins G and L using roadmaps computed with energy thresholds $E_{\mathrm{min}} = 50\,000$ kJ mol^{-1} and $E_{\mathrm{max}} = 70\,000$ kJ mol^{-1}. For each combination of contact type (all or hydrophobic) and number of contacts (first $x\%$ to form), we show the percentage of pathways with a particular secondary structure formation order. Recall that β-hairpin 2 ($\beta3$–$\beta4$) forms first in Protein G and β-hairpin 1 ($\beta1$–$\beta2$) forms first in Protein L.

Name	Contacts	Secondary structure formation order	20%	40%	60%	80%	100%
Protein G	All	α, $\beta3$–$\beta4$, $\beta1$–$\beta2$, $\beta1$–$\beta4$	76	66	77	55	58
		α, $\beta1$–$\beta2$, $\beta3$–$\beta4$, $\beta1$–$\beta4$	23	34	23	45	42
	Hydrophobic	α, $\beta3$–$\beta4$, $\beta1$–$\beta2$, $\beta1$–$\beta4$	85	78	77	62	67
		α, $\beta3$–$\beta4$, $\beta1$–$\beta4$, $\beta1$–$\beta2$	11	11	9	8	8
		α, $\beta1$–$\beta2$, $\beta3$–$\beta4$, $\beta1$–$\beta4$	4	10	14	29	24
Protein L	All	α, $\beta1$–$\beta2$, $\beta3$–$\beta4$, $\beta1$–$\beta4$	67	76	78	78	92
		α, $\beta1$–$\beta2$, $\beta1$–$\beta4$, $\beta3$–$\beta4$	15	4	4	4	4
		α, $\beta3$–$\beta4$, $\beta1$–$\beta2$, $\beta1$–$\beta4$	19	20	18	18	4
	Hydrophobic	α, $\beta1$–$\beta2$, $\beta3$–$\beta4$, $\beta1$–$\beta4$	54	65	74	73	86
		α, $\beta1$–$\beta2$, $\beta1$–$\beta4$, $\beta3$–$\beta4$	9	3	3	2	2
		α, $\beta3$–$\beta4$, $\beta1$–$\beta2$, $\beta1$–$\beta4$	36	32	23	26	13

Potential energy calculations

We have tested our technique using two potential energy functions.

The first is a coarse potential similar to [20]. We use a step function approximation of the van der Waals potential component and model side chains as spheres with zero DOF. If any two side chain spheres are too close (i.e. less than 2.4 Å during node generation and 1.0 Å during node connection), a very high potential is returned. Otherwise, the potential is

$$U_{\mathrm{tot}} = \sum_{\mathrm{restraints}} K_{\mathrm{d}}\{[(d_i - d_0)^2 + d_{\mathrm{c}}^2]^{1/2} - d_{\mathrm{c}}\} + E_{\mathrm{hp}} \tag{1}$$

The first term represents constraints favoring known secondary structure through main-chain hydrogen bonds and disulphide bonds. The second term is the hydrophobic effect. It takes 8–10 hours to build a reasonable roadmap containing thousands of folding pathways for a small protein (60–80 amino acids) with this potential.

The second potential is the Effective Energy Function 1 all-atoms potential [47]. The running time increases to two weeks with this much finer potential. Thus, it is imperative that we have a parallel solution to make experiments on larger proteins with increased accuracy feasible.

STAPL

STAPL is a framework for parallel C++ code [10–12]. Its core is a library of ISO Standard C++ components with interfaces similar to the (sequential) ISO C++ standard library [13]. STAPL offers the parallel system programmer a shared object view of the data space. The objects are distributed across the memory hierarchy which can be shared and/or distributed address spaces. Internal STAPL mechanisms assure an automatic translation from one space to another, presenting to the less experienced user a flat, unified data space. For more experienced users, the local/remote distinction of accesses can be exposed and performance enhanced. STAPL supports the single process multiple data (SPMD) model of parallelism with essentially the same consistency model as OpenMP. To exploit large systems such as IBM's BlueGene/L, STAPL allows for (recursive) nested parallelism (as in NESL [48]). Because performance of parallel algorithms is sensitive to the system architecture, to application data, and to run-time conditions, STAPL is designed to continually adapt to the system and the data at all levels—from selecting the most appropriate algorithmic implementation to balancing communication granularity with latency, etc.

The STAPL infrastructure consists of platform-independent and platform-dependent components. These are revealed to the programmer at an appropriate level of detail through a hierarchy of abstract interfaces. The platform independent components include the core parallel library, a view of a generic parallel/distributed machine, and an abstract interface to the communication library and run-time system.

The core library consists of parallel algorithms (`pAlgorithms`) and distributed data structures (`pContainers`). A `pContainer` is the parallel equivalent of a STL container. Its data is distributed, but the programmer is offered a shared object view. The `pContainer` distribution can be user specified or computed automatically. A `pAlgorithm` is the parallel equivalent of an STL algorithm. STAPL binds `pAlgorithms` to `pContainers` with a `pRange` class which supports random access to distributed `pContainer` data. STAPL's ARMI (Adaptive Remote Method Invocation) communication library [12] uses the remote method invocation (RMI) communication abstraction that assures mutual exclusion at the destination, but hides the lower-level implementation (e.g. MPI, OpenMP).

One `pContainer` provided by STAPL is the `pGraph`, a parallel and distributed counterpart of a sequential graph data structure which is also provided with STAPL. `pGraph`, as all `pContainers` in STAPL, provides a shared object view of the data stored inside. Every vertex is associated with a unique identifier which STAPL uses to decide whether a specific method should be called locally or on a remote thread. The `pGraph` interface is the same as the sequential graph interface provided by STL. This is a great help when parallelizing an existing sequential implementation. For our protein-folding application, using `pGraph` to store the roadmap yields a straightforward parallelization of the initial sequential code which was implemented using STL's sequential graph.

The `pGraph` provides methods to insert and delete vertices and edges in parallel, methods to access the data associated with vertices and edges, and a collection of algorithms that is continuously expanding (e.g. DFS, BFS, shortest paths, connected components, strongly connected components, etc.). The `pGraph` employs an object-oriented design where different graph characteristics are enclosed in modules that can be instantiated depending on the application's needs. These characteristics include directed or undirected edges, weighted or unweighted edges, and unique or multiple edges allowed between vertices. For example, specifying a directed, weighted `pGraph` with unique edges can be expressed in STAPL as follows:

```
stapl::pGraph<Directed,Weighted,NonMultiple,VertexData,EdgeData> pg;
```

where `VertexData` and `EdgeData` are data types for the vertices and edges, respectively.

```
Input: The protein's native state $q_{\text{native}}$.
Output: A graph $R$ of the protein's potential energy landscape.
  Generate $n$ nodes biased toward $q_{\text{native}}$ and add them to $R$.
  for each $q \in R$ do
    Let $N_k(q)$ be the $k$ closest neighbors of $q$.
    for each $q' \in N_k(q)$ do
      if the local planner can find a path between $q$ and $q'$ then
        Add the edge $(q, q')$ to $R$.
      end if
    end for
  end for
  return $R$.
```

Algorithm 1. Sequential PRM-based protein-folding algorithm.

The STAPL run-time system (RTS) is a collection of platform specific components that needs to be adapted whenever STAPL is ported to a new system. Here, remote references to shared objects are detected and the generic RMI is translated to MPI, OpenMP, pthreads, or the native communication primitives. The RTS also includes memory management methods that are tailored to the hardware, e.g. processor aware allocation, mapping of virtual processor to physical processor, etc., and other low-level optimizations.

PARALLEL PROTEIN FOLDING

The sequential PRM-based protein folding algorithm is outlined in Algorithm 1. The dominating operation is the potential energy calculation. This is performed once for each of the n roadmap nodes. In addition, for each node q and each of its k nearest neighbors q', all the edges (q, q') must be checked. To check the edge (q, q'), the potential energy calculation is called a number of times proportional to the C-space distance between q and q'. Assuming that q and q' are an average distance d apart, roadmap construction requires $O(nkd)$ potential energy calculations, most of which are independent.

This observation gives rise to a straightforward parallel version described in Algorithm 2. First, each processor generates n/p nodes biased toward the native state and adds them to the roadmap R, which is stored in a `pGraph`. Then, the sequential `for` loop in line 2 of Algorithm 1 is replaced with a parallel `for` loop. This is similar to the parallel PRM algorithm in [9]. Each processor checks the candidate edges for its nodes in parallel. This requires little communication compared with the computation used by the potential energy calculations needed to evaluate the edge.

Since the roadmap is stored in a `pGraph`, STAPL automatically manages the communication between processors for the user. For node generation, each processor p_i (for $0 \leq i < p$) adds n/p nodes to the `pGraph` roadmap in parallel. Then during the node connection phase, each processor computes the k nearest neighbors between each of its nodes and the nodes in the entire roadmap. Each processor then checks its candidate edges and adds the successful ones to the roadmap in parallel. STAPL masks all the communication from the user.

```
Input: The protein's native state $q_{\text{native}}$.
Output: A pGraph $R$ of the protein's potential energy landscape.
  Each processor generates $n/p$ nodes biased toward $q_{\text{native}}$ and adds them to $R$.
  for each $q \in R$ par do
    Let $N_k(q)$ be the $k$ closest neighbors of $q$.
    for each $q' \in N_k(q)$ do
      if the local planner can find a path between $q$ and $q'$ then
        Add the edge $(q, q')$ to $R$.
      end if
    end for
  end for
  return $R$.
```

Algorithm 2. Parallel PRM-based protein-folding algorithm using p processors.

Table III. Characteristics of the three proteins studied.

Name	PDB ID	Description	Size	Structure
A	1BDD	Protein A, B domain	60	3α
G	1GB1	Protein G, B1 domain	56	$1\alpha + 4\beta$
CTXIII	2CRT	Cardiotoxin III	60	5β

EXPERIMENTAL RESULTS AND DISCUSSION

In this section, we study the performance gains of the parallel protein-folding code using STAPL over the original sequential code. We compare results on three small proteins with differing structures previously studied in [4] (see Table III). Protein A consists of three alpha helices, CTXIII consists of five beta strands, and G has one alpha helix and four beta strands.

Experimental setup

We study the performance on a range of systems: a small dedicated homogeneous Linux cluster (Linux Cluster A), a distributed shared memory machine (SGI Altix), a dedicated Linux cluster at Lawrence Livermore National Laboratory (MCR), and IBM's Linux-based BlueGene/L.

Linux Cluster A consists of four boards, each of which has two processors and 2 GB RAM. Two boards have 1 GHz processors with 256 KB caches, and two boards have 1.1 GHz processors with 512 KB caches. They are connected with a Gbit dedicated Ethernet switch.

The SGI Altix 3700 is a distributed shared-memory machine in the Texas A&M University Supercomputing facility. It contains 32 nodes, each with two pairs of 1.3 GHz 64-bit processors, and 256 GB RAM.

MCR (Multiprogrammatic Capability Cluster) is a large, dedicated Linux cluster at the Lawrence Livermore National Laboratory. It has 1152 nodes with two 2.4 GHz processors and 4 GB RAM each. They are connected with a Gbit Ethernet switch.

BlueGene/L is a scalable massively parallel 180 Teraflop machine which will have up to 65 536 compute nodes, each with 256 MB of memory, configured as a $64 \times 32 \times 32$ three-dimensional torus. Each node has a single ASIC and 256 MB of memory.

We used an eight-processor configuration of Linux Cluster A and 32 processor configurations of the SGI Altix, MCR and BlueGene/L. Although MCR and BlueGene/L are large machines, they were both under development and we had only limited access to them, and hence here we present results only for small partitions of the machines that have comparable size with the Altix.

The same program, with no user code modification, was run on all systems. The results presented here use the coarser potential energy computation; using the all-atoms potential yields better performance gains since it requires significantly more computation and essentially the same communication.

The same proteins were studied on all four systems. However, the results presented here on the various systems do have different input sizes. This is due to our limited access to some of the systems, but mainly also so that we can illustrate the different performance levels of the algorithm. In particular, for Linux Cluster A and BlueGene/L we attempt to build a roadmap with 750 nodes, for the SGI Altix we attempt to build a roadmap with 1500 nodes, and for MCR the objective is a roadmap with 2000 nodes. Hence, the roadmaps for Linux Cluster A and BlueGene/L are significantly smaller than those for the Altix or MCR. We note that all of these problem sizes are quite small—in practice, a map for one of these proteins should have between 5000 and 10 000 samples.

Even when a particular number of nodes is requested, the method may not produce precisely that many because the sampling is randomized and does not always succeed. The results presented for a single protein and system are normalized with respect to the number of nodes produced with just one thread for that protein. In particular, if n_p and t_p are the number of nodes produced and the time required to produce them for a particular protein with p threads, then the normalized time $\overline{t_p}$ for p threads is

$$\overline{t_p} = t_p \left(\frac{n_1}{n_p}\right)$$

Experimental results

We now review the performance of the algorithm on the four systems. Figures 2, 3, 4 and 5 show results for Linux Cluster A, MCR, the SGI Altix, and BlueGene/L, respectively. In each figure, part (a) gives speedups of the complete roadmap construction algorithm over the sequential version for each of the three proteins studied, and part (b) shows the speedups for the node generation phase and the node connection phase of the roadmap construction algorithm for one of the proteins.

On all systems, we note in part (a) that the speedups obtained are similar for all proteins studied, indicating that performance is not dependent on the protein's structure. The variance that is visible in the plots is in fact due to the random nature of the process. To illustrate this variance, we have presented results from a single run rather than averaging over many executions. We also note that the speedups on BlueGene/L are somewhat less than on the Altix and MCR. This is because, as mentioned above,

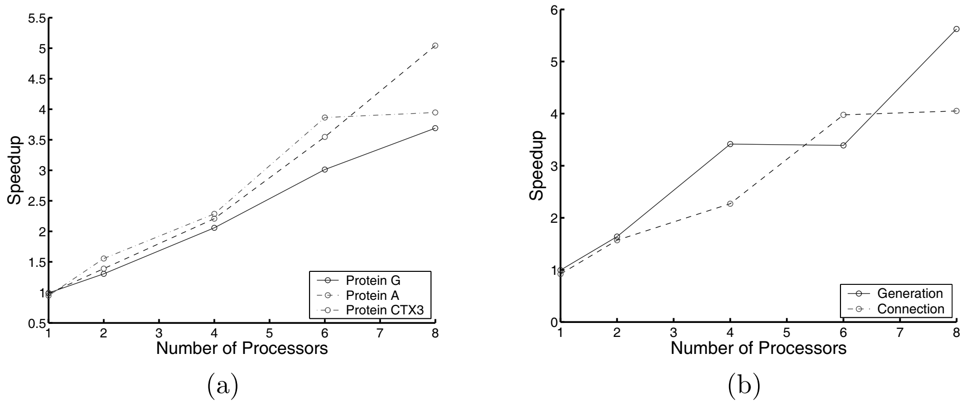

Figure 2. Linux Cluster A speedups: (a) total time for each protein; (b) each construction phase for Protein CTXIII.

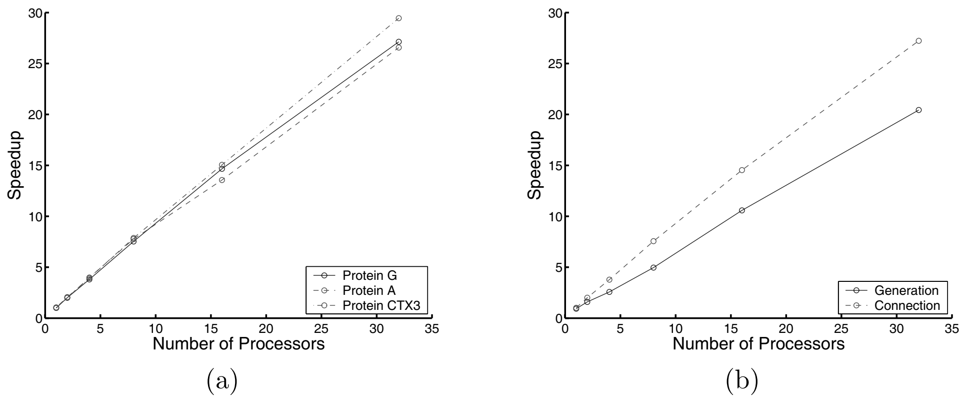

Figure 3. Altix speedup: (a) total time for each protein; (b) each construction phase for Protein A.

smaller roadmaps (and therefore less computation per thread) were produced on BlueGene/L (750 total samples were requested) than on the Altix and MCR (1500 and 2000 samples requested, respectively).

When we examine the different phases of the algorithm in part (b), we note that in all systems, except Linux Cluster A, the speedups for the node generation (sampling) phase are lower than for the connection phase of the roadmap construction. This is because there is significantly more computation required in the node connection step than in the node generation step, and hence there is relatively more communication in the node generation step. Nevertheless, when comparing part (a) and part (b) for each system, we note that the connection step is the dominant component of the algorithm's performance.

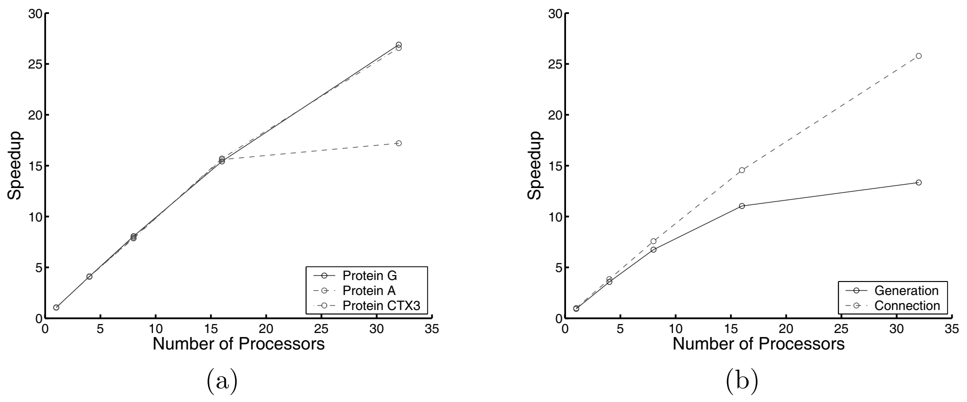

Figure 4. MCR speedup: (a) total time for each protein; (b) each construction phase for Protein G.

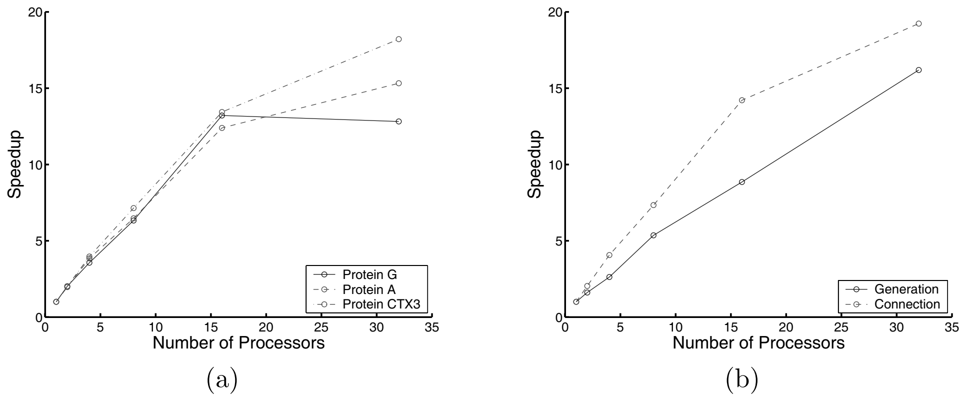

Figure 5. BlueGene/L speedup: (a) total time for each protein; (b) each construction phase for Protein CTXIII.

Moreover, when building larger maps, the number of samples generated per thread would be greater and the node generation step would be expected to scale better.

CONCLUSION

In this paper we described a parallel implementation of our sequential technique that can compute maps of a protein's energy landscape containing thousands of folding pathways in a relatively short amount of time. With STAPL, we were able to easily parallelize our sequential code and, moreover were able to obtain scalable speedups on a variety of platforms with no user code modification. We plan to use our parallel protein-folding technique to study larger, more complex proteins with a higher degree of accuracy.

ACKNOWLEDGEMENTS

This research was supported in part by NSF grants ACI-9872126, EIA-9975018, EIA-0103742, EIA-9805823, ACR-0081510, ACR-0113971, CCR-0113974, EIA-9810937, EIA-0079874, IIS-0118097, by DOE grant B520713, and by the Texas Higher Education Coordinating Board grant ATP-000512-0261-2001. Thomas was supported in part by a NSF Graduate Research Fellowship. Portions of this work were done while Amato and Rauchwerger were Academic Visitors at IBM T. J. Watson Research Center in Yorktown Heights, NY, U.S.A.

REFERENCES

1. Levitt M, Gerstein M, Huang E, Subbiah S, Tsai J. Protein folding: The endgame. *Annual Review of Biochemistry* 1997; **66**:549–579.
2. Reeke GN Jr. Protein folding: Computational approaches to an exponential-time problem. *Annual Review of Computer Science* 1988; **3**:59–84.
3. Lansbury PT. Evolution of amyloid: What normal protein folding may tell us about fibrillogenesis and disease. *Proceedings of the National Academy of Science of the U.S.A.* 1999; **96**(7):3342–3344.
4. Amato NM, Song G. Using motion planning to study protein folding pathways. *Journal of the Computational Biology* 2002; **9**(2):149–168.
5. Amato NM, Dill KA, Song G. Using motion planning to map protein folding landscapes and analyze folding kinetics of known native structures. *Journal of Computational Biology* 2003; **10**(3–4):239–256.
6. Song G, Thomas SL, Dill KA, Scholtz JM, Amato NM. A path planning-based study of protein folding with a case study of hairpin formation in protein G and L. *Proceedings of the Pacific Symposium of Biocomputing (PSB)*, 2003; 240–251.
7. Kavraki LE, Svestka P, Latombe JC, Overmars MH. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation* 1996; **12**(4):566–580.
8. Li R, Woodward C. The hydrogen exchange core and protein folding. *Protein Science* 1999; **8**(8):1571–1591.
9. Amato NM, Dale LK. Probabilistic roadmap methods are embarrassingly parallel. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 1999; 688–694.
10. An P, Jula A, Rus S, Saunders S, Smith T, Tanase G, Thomas N, Amato N, Rauchwerger L. STAPL: An adaptive, generic parallel programming library for C++. *Proceedings of the 14th International Workshop on Languages and Compilers for Parallel Computing (LCPC)*, Cumberland Falls, KY, August 2001.
11. Rauchwerger L, Arzu F, Ouchi K. Standard Templates Adaptive Parallel Library. *Proceedings of the 4th International Workshop on Languages, Compilers and Run-Time Systems for Scalable Computers (LCR)*, Pittsburgh, PA, May 1998.
12. Saunders S, Rauchwerger L. ARMI: An adaptive, platform independent communication library. *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*, San Diego, CA, June 2003.
13. Musser D, Derge G, Saini A. *STL Tutorial and Reference Guide* (2nd edn). Addison-Wesley: Reading, MA, 2001.
14. Bryngelson JD, Onuchic JN, Socci ND, Wolynes PG. Funnels, pathways, and the energy landscape of protein folding: A synthesis. *Proteins: Structural and Functional Genetics* 1995; **21**:167–195.
15. Levitt M, Warshel A. Computer simulation of protein folding. *Nature* 1975; **253**:694–698.
16. Sun S, Thomas PD, Dill KA. A simple protein folding algorithm using a binary code and secondary structure constraints. *Protein Engineering* 1995; **8**(8):769–778.
17. Daggett V, Levitt M. Realistic simulation of naive-protein dynamics in solution and beyond. *Annual Review of Biophysics and Biomolecular Structures* 1993; **22**:353–380.
18. Duan Y, Kollman PA. Pathways to a protein folding intermediate observed in a 1-microsecond simulation in aqueous solution. *Science* 1998; **282**:740–744.
19. Haile JM. *Molecular Dynamics Simulation: Elementary Methods*. Wiley: New York, 1992.
20. Levitt M. Protein folding by restrained energy minimization and molecular dynamics. *Journal of Molecular Biology* 1983; **170**:723–764.
21. Covell DG. Folding protein α-carbon chains into compact forms by Monte Carlo methods. *Proteins: Structural and Functional Genetics* 1992; **14**(4):409–420.

22. Kolinski A, Skolnick J. Monte Carlo simulations of protein folding. *Proteins: Structural and Functional Genetics* 1994; **18**(3):338–352.
23. Bowie JU, Eisenberg D. An evolutionary approach to folding small α-helical proteins that uses sequence information and an empirical guiding fitness function. *Proceedings of the National Academy of Science of the U.S.A.* 1994; **91**(10):4436–4440.
24. Sun S. Reduced representation model of protein structure prediction: Statistical potential and genetic algorithms. *Protein Science* 1993; **2**(5):762–785.
25. Alm E, Baker D. Prediction of protein-folding mechanisms from free-energy landscapes derived from native structures. *Proceedings of the National Academy of Science of the U.S.A.* 1999; **96**(20):11 305–11 310.
26. Baker D. A surprising simplicity to protein folding. *Nature* 2000; **405**:39–42.
27. Muñoz V, Henry ER, Hoferichter J, Eaton WA. A statistical mechanical model for β-hairpin kinetics. *Proceedings of the National Academy of Science of the U.S.A.* 1998; **95**:5872–5879.
28. Apaydin MS, Brutlag DL, Guestrin C, Hsu D, Latombe J-C. Stochastic roadmap simulation: An efficient representation and algorithm for analyzing molecular motion. *Proceedings of the International Conference on Computational Molecular Biology (RECOMB)*, 2002; 12–21.
29. Apaydin MC, Singh AP, Brutlag DL, Latombe J-C. Capturing molecular energy landscapes with probabilistic conformational roadmaps. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2001; 932–939.
30. Latombe J-C. *Robot Motion Planning*. Kluwer Academic: Boston, MA, 1991.
31. Lozano-Pérez T. Spatial planning: A configuration space approach. *IEEE Transactions on Computing* 1983; **32**:108–120.
32. Crowley MF, Darden TA, Cheatham TE, Deerfield DW. Adventures in improving the scaling and accuracy of a parallel molecular dynamics program. *Journal of Supercomputing* 1997; **11**:255–278.
33. Matthey T, Ko A, Izaguirre JA. *PROTOMOL: A Molecular Dynamics Research Framework for Algorithmic Development* (*Lecture Notes in Computer Science*, vol. 2659). Springer: Berlin, 2003; 50–59.
34. Weiner PK, Kollman PA. Amber: Assisted model building with energy renement, a general program for modeling molecules and their interactions. *Journal of Computational Chemistry* 1981; **2**:287–303.
35. Kalé L, Skeel R, Bhandarkar M, Brunner R, Gursoy A, Krawetz N, Phillips J, Shinozaki A, Varadarajan K, Schulten K. NAMD2: Greater scalability for parallel molecular dynamics. *Journal of Computational Physics* 1999; **151**:283–312.
36. Nelson MT, Humphrey W, Gursoy A, Dalke A, Kalé LV, Skeel RD, Schulten K. NAMD: A parallel, object oriented molecular dynamics program. *International Journal of Supercomputer Applications* 1996; **10**:251–268.
37. Pande VS *et al.* Atomistic protein folding simulations on the submillisecond time scale using worldwide distributed computing. *Biopolymer* 2003; **68**(1):91–109.
38. Shirts M, Pande VS. Screen savers of the world unite. *Science* 2000; **290**:1903–1904.
39. Snow CD, Nguyen N, Pande VS, Gruebele M. Absolute comparison of simulated and experimental protein-folding dynamics. *Nature* 2002; **420**:102–106.
40. Zagrovic B, Snow CD, Shirts MR, Pande VS. Simulation of folding of a small alpha-helical protein in atomistic detail using worldwide-distributed computing. *Journal of Molecular Biology* 2002; **323**:927–937.
41. Lozano-Pérez T, O'Donnell P. Parallel robot motion planning. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 1991; 1000–1007.
42. Challou DJ, Gini M, Kumar V. Parallel search algorithms for robot motion planning. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 1993; 46–51.
43. Challou D, Boley D, Gini M, Kumar V. A parallel formulation of informed randomized search for robot motion planning problems. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 1995; 709–714.
44. Barraquand J, Latombe JC. Robot motion planning: A distributed representation approach. *International Journal of Robotic Research* 1991; **10**(6):628–649.
45. Sternberg MJ. *Protein Structure Prediction*. OIRL Press at Oxford University Press: Oxford, 1996.
46. Fiebig KM, Dill KA. Protein core assembly processes. *Journal of Chemical Physics* 1993; **98**(4):3475–3487.
47. Lazaridis T, Karplus M. Effective energy function for proteins in solution. *Proteins* 1999; **35**:133–152. Available at: http://mingus.sci.ccny.cuny.edu/server/.
48. Blelloch G. NESL: A nested data-parallel language. *Technical Report CMU-CS-93-129*, Carnegie Mellon University, April 1993.