Parallel Protein Folding with STAPL[*]

Shawna Thomas   Nancy M. Amato
Dept. of Computer Science
Texas A&M University
{sthomas, amato}@cs.tamu.edu

Abstract

The protein folding problem is to study how a protein dynamically folds to its so-called native state – an energetically stable, three-dimensional configuration. 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. In our group, 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 *(the 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 without user code modification. We show performance gains on two systems: a dedicated Linux cluster and an extremely heterogeneous multiuser Linux cluster.*

1 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 3D structure is important because the protein's function is related to it. 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 [23]. In this work, we assume the native structure is known and concentrate on studying the protein folding mechanisms. 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, 3, 34], 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) [21], that have proven effective on a wide range of applications 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 [28]. 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 [34].

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 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" [2]. 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) [33, 5] to par-

[*]This research supported in part by NSF Grants ACI-9872126, EIA-9975018, EIA-0103742, EIA-9805823, ACR-0081510, ACR-0113971, CCR-0113974, EIA-9810937, EIA-0079874, by the DOE ASCI ASAP program, and by the Texas Higher Education Coordinating Board grant ATP-000512-0261-2001. Thomas supported in part by an NSF Graduate Research Fellowship.

Table 1. A comparison of protein folding models.

Comparison of Models for Protein Folding					
Approach	**Folding Landscape**	**# Paths Produced**	**Path Quality**	**Compute Time**	**Native 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	Approx	Fast	Yes
Lattice Model	Not used on real proteins				

allelize 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) [32] 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 two systems: a homogeneous dedicated Linux cluster and a heterogeneous non-dedicated Linux cluster. We also demonstrate how STAPL enables one to run the same parallel application on several different platforms without modifying user code.

2 Related Work

Protein folding. Several computational approaches have been applied to the protein folding problem, see Table 1. These include lattice models [12], energy minimization [27, 37], molecular dynamics [26, 19, 16, 17], Monte Carlo methods [15, 22], and genetic algorithms [11, 36]. Molecular dynamics and Monte Carlo methods provide a single, high quality folding pathway, but each run is computationally intensive. Statistical mechanical models [31, 1, 8], while computationally 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.

Apaydin et al. [7, 6] have also used PRM-based techniques to study protein folding, however their work differs from ours in several aspects. First, they model the protein at a much coarser level considering each secondary structure to be rigid. Second, while our focus is on studying the folding process, their focus has been to compare the PRM approach with methods such as Monte Carlo simulation.

Probabilistic Roadmap Method. Given a description of the environment and a movable object, the motion planning problem is to find a valid path for the movable object from a start configuration to a goal configuration. The probabilistic roadmap method (PRM) [21] 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 [29]. 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. This roadmap can then be used to answer different queries or 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 been restricted to molecular dynamics simulations. Peter Kollman pioneered work in this area with AMBER [39]. NAMD [20] 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 [24] 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.

There has been some work on parallel motion planning. Lozano-Perez and O'Donnell [30] developed a parallel algorithm for computing a discretized C-space for the first 3 links of a 6 dof articulated linkage. Challou et al. [14, 13] parallelize the Randomized Path Planner [9], a randomized potential field method. Finally, [2] shows that PRMs are "embarrassingly parallel" and gives impressive speedups.

3 Protein Folding using PRMs

In previous work, we successfully applied the PRM framework to study protein folding pathways [4, 3, 34]. The protein is modeled as an articulated linkage. Using a stan-

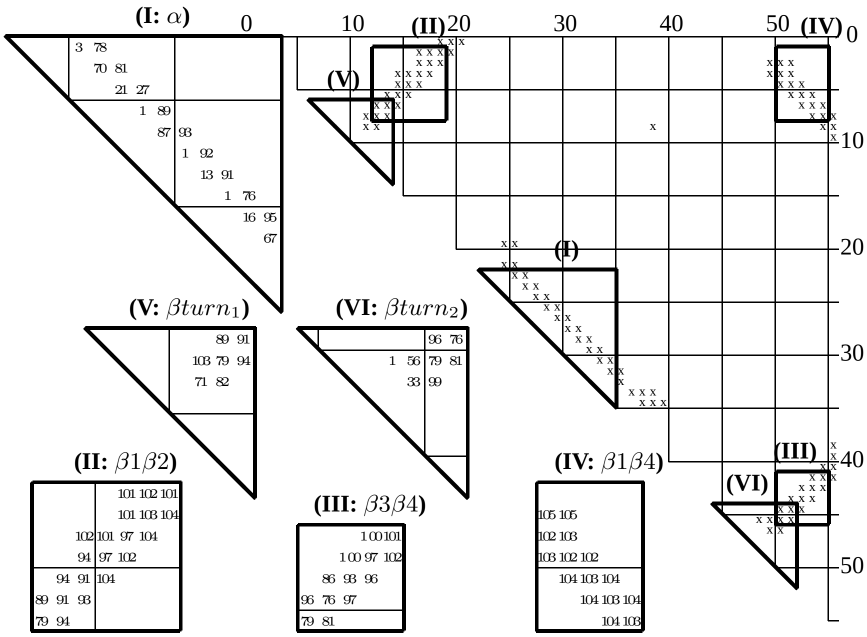

Figure 1. Timed Contact Map for Protein G. The full contact matrix (right) and blow-ups (left) showing the time steps when contacts appear on a path. The blow-ups: I: α helix, II: $\beta 1\beta 2$, III: $\beta 3\beta 4$, IV: $\beta 1\beta 4$, V: turn 1 ($\beta 1\beta 2$), and VI: turn 2 ($\beta 3\beta 4$).

dard modeling assumption for proteins that bond angles and bond lengths are fixed [35], the only dof in our model are the backbone's phi and psi torsional angles. These are modeled as revolute joints taking values $[2, \pi)$.

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:

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

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 ($\sim$60 amino acids) and validated our pathways by comparing the secondary structure formation order with known experimental results [28] 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 [34].

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 7Å 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 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 [18]. 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

Table 2. Comparison of analysis techniques for proteins G and L using roadmaps computed with energy thresholds $E_{\min} = 50,000$ kJ/mol and $E_{\max} = 70,000$ kJ/mol. 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 (β3-β4) forms first in protein G and β-hairpin 1 (β1-β2) forms first in protein L.

Name	Contacts	SS Formation Order	20%	40%	60%	80%	100%
Protein G	all	α, β3-β4, β1-β2, β1-β4	76	66	77	55	58
		α, β1-β2, β3-β4, β1-β4	23	34	23	45	42
	hydrophobic	α, β3-β4, β1-β2, β1-β4	85	78	77	62	67
		α, β3-β4, β1-β4, β1-β2	11	11	9	8	8
		α, β1-β2, β3-β4, β1-β4	4	10	14	29	24
Protein L	all	α, β1-β2, β3-β4, β1-β4	67	76	78	78	92
		α, β1-β2, β1-β4, β3-β4	15	4	4	4	4
		α, β3-β4, β1-β2, β1-β4	19	20	18	18	4
	hydrophobic	α, β1-β2, β3-β4, β1-β4	54	65	74	73	86
		α, β1-β2, β1-β4, β3-β4	9	3	3	2	2
		α, β3-β4, β1-β2, β1-β4	36	32	23	26	13

order occurring. Table 2 shows secondary structure formation order results for Proteins G and L [34]. Our results were able to identify the different folding behaviors of the two structurally similar proteins.

Potential energy calculations. We have tested two potential energy calculations with our technique.

The first is a coarse potential similar to [26]. 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 2.4Å during node generation and 1.0Å during node connection), a very high potential is returned. Otherwise, the potential is:

$$U_{tot} = \sum_{restraints} K_d\{[(d_i - d_0)^2 + d_c^2]^{1/2} - d_c\} + E_{hp} \quad (1)$$

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 with this potential.

The second potential is the Effective Energy Function 1 all-atoms potential [25]. The running time increases to 2 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.

4 STAPL

STAPL (the Standard Template Adaptive Parallel Library) is a framework for parallel C++ code [5, 33]. Its core is a library of ISO Standard C++ components with interfaces similar to the (sequential) ISO C++ standard library [32]. 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, uniform memory access-like, unified data space. For more experienced users the local/remote distinction of accesses can be exposed and performance enhanced. STAPL supports the SPMD model of parallelism with essentially the same consistency model as OpenMP. To exploit large systems like the DOE machines and IBM's BlueGene/L, STAPL allows for (recursive) nested parallelism (as in NESL [10]). Because performance of parallel algorithms is sensitive to 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 an STL container. Its data is distributed but offers 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. Communication and synchronization use the remote method invocation (RMI) communication abstraction that assures mutual exclusion at the destination but hides the lower level implementation (e.g., MPI, OpenMP).

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 native communication primitives. The memory management is also tailored to the specifics of the hardware, e.g., processor aware allocation, mapping of virtual processor to physical processor, etc., and other low-level optimizations are performed here.

Two `pContainers` provided by STAPL are `pVector` [5] and `pArray` [38]. They have the same interface as their sequential counterparts — `pVector` is a parallel/distributed version of the STL vector, and `pArray` is a parallel/distributed version of the valarray. `pVector` and `pArray` provide (semi-) random access to a sequence of elements in shared or distributed memory. `pArray` has a fixed number of elements, but `pVector` may grow or shrink during execution. Both support data redistribution.

5 Parallel Protein Folding

The sequential PRM-based protein folding algorithm is given by Algorithm 5.1. 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 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 that is 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.

Algorithm 5.1 Sequential PRM-based protein folding algorithm.

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

This observation leads to a straightforward parallel version replacing the sequential `for` loop in line 2 of Algorithm 5.1 with a parallel `for` loop. This is similar to the parallel PRM algorithm in [2]. Nodes are divided among the processors. Each processor checks the candidate edges for its nodes in parallel. This requires little communication.

Although it is possible to further parallelize the algorithm by generating roadmap nodes in parallel, we refrain in this initial study for two reasons: (i) 99% of the computation time is spent during node connection, very little is spent generating nodes and (ii) our biased sampling method would require more processor communication as each generated node is not independent of the others. We find that we get significant speedups without parallelizing the node generation phase, but we do plan to parallelize it in the future.

We use STAPL to manage communication between processors during node connection. We study two implementations: one with `pVector` and one with `pArray`. During node connection, each processor p_i (for $0 \leq i < p$) adds $k(n/p)$ candidate edges to the `pContainer` indexed beginning at $i \times k(n/p)$. The `pContainer` is distributed so elements accessed by p_i are local to p_i. Each processor then checks its candidate edges and records the results in the `pContainer`. Finally, the valid edges in the `pContainer` are sequentially added to the roadmap. STAPL masks all the communication from the user.

6 Experimental Results and Discussion

In this section, we study the performance gains of the parallel protein folding code using STAPL over the sequential code. We compare results on three small proteins with differing structures previously studied in [4] (see Table 3).

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

Table 3. Characteristics of the three proteins studied.

6.1 Experimental Setup

We study the performance on two systems: a dedicated Linux cluster (Linux Cluster A) and a non-dedicated extremely heterogeneous Linux cluster (Linux Cluster B). Linux Cluster A consists of 4 boards. Each board has 2 processors and 2GBs RAM. Two boards have 1GHz processors with 256KB caches, and two boards have 1.1GHz processors with 512 KB caches. They are connected with a

Gbit dedicated Ethernet switch. Linux Cluster B has 10 systems; the individual system specs are given in Table 4. The systems and the interconnects are not dedicated. Because there is a large discrepancy between some of the systems (e.g., systems 9 and 10 as compared with the others), we attempt to load balance by executing multiple processes on the other systems when running 12 and 16-way jobs. The code was compiled without any optimizations. Results use the coarser potential. Using the all-atoms potential yields better performance gains.

System ID	Processor Speed (GHz)	Cache Size (KB)	Memory (MBs)
1–4	2.8	512	512
5	2.8	512	1024
6	2.4	512	640
7–8	1.8	256	512
9–10	0.5	512	512

Table 4. Individual system specs for Linux Cluster B.

6.2 Results

Performance on Linux Cluster A. Here we study the performance gains on a dedicated Linux Cluster up to 8 processors. Figure 2 gives speedups over the sequential version for `pVector`; the results for `pArray` are similar. The parallel code scales well and greatly reduces the total running time. Performance gains are similar for all proteins studied indicating that performance is not dependent on the protein's structure.

Figure 3 displays the speedups for the connection phase. Performance is only marginally better confirming our statement that most of the running time is spent in the connection phase and little is spent during node generation.

Performance on Linux Cluster B. Here we look at the performance gains on a non-dedicated Linux Cluster. No user code modification was required. Figure 4 gives the speedups for `pVector`; the results for `pArray` are similar. The code scales well up to 8 processors, then performance tapers off. This is caused by two factors. First, each additional system is slower than previous systems. Performance degrades because each system is given the same amount of work. A more sophisticated load-balancing algorithm would help alleviate this. Second, the system network is shared by the entire department. This increases communication cost. This decreases performance stability between Cluster B and Cluster A.

Figure 5 shows speedups for the connection phase. They are similar to those for the entire program execution.

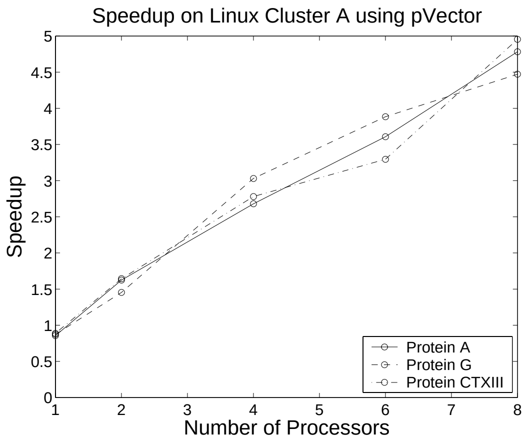

Figure 2. Speedup on Linux Cluster A using `pVector`.

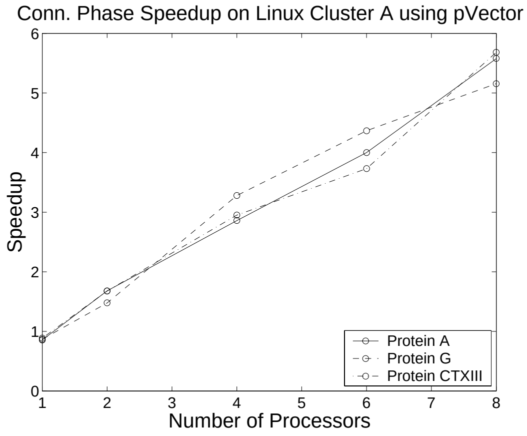

Figure 3. Speedup of the connection phase only on Linux Cluster A using `pVector`

7 Conclusion

We presented a parallel protein folding algorithm 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 to obtain scalable speedups. STAPL also enabled portability across multiple platforms with no user code modification. We expect that our parallel protein folding technique will enable the study of larger, more complex proteins with a higher degree of accuracy.

In the future, we plan to study the performance gains using `pGraph` to store the roadmap. `pGraph`, a STAPL `pContainer` currently under development, is a distributed graph data structure. It allows parallel insertion of nodes and edges and will enable a more straightforward par-

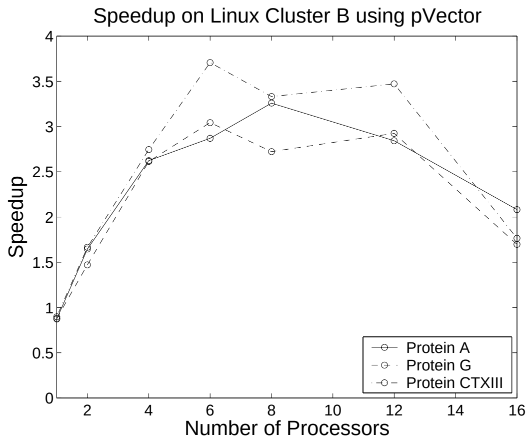

Figure 4. Speedup on Linux Cluster B using `pVector`.

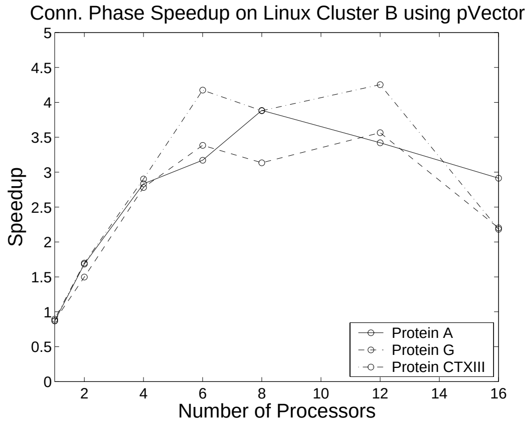

Figure 5. Speedup of the connection phase only on Linux Cluster B using `pVector`.

allelization of the code.

`pGraph` will also reduce some extra overhead incurred by the current parallel implementation. In the sequential code, an edge is only validated if it not already in the roadmap. Thus, valid edges are not checked more than once. There is a nontrivial subset of repeated edges using the k-closest connection strategy that are checked multiple times in the current parallel implementation. `pGraph` will eliminate this problem. This will eliminate most validation redundancy.

References

[1] E. Alm and D. Baker. Prediction of protein-folding mechanisms from free-energy landscapes derived from native structures. *Proc. Natl. Acad. Sci. USA*, 96(20):11305–11310, 1999.

[2] N. M. Amato and L. K. Dale. Probabilistic roadmap methods are embarrassingly parallel. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 688–694, 1999.

[3] N. M. Amato, K. A. Dill, and G. Song. Using motion planning to map protein folding landscapes and analyze folding kinetics of known native structures. *J. Comput. Biol.*, 2003. To appear. Special issue of Int. Conf. Comput. Molecular Biology (RECOMB) 2002.

[4] N. M. Amato and G. Song. Using motion planning to study protein folding pathways. *J. Comput. Biol.*, 9(2):149–168, 2002. Special issue of Int. Conf. Comput. Molecular Biology (RECOMB) 2001.

[5] P. An, A. Jula, S. Rus, S. Saunders, T. Smith, G. Tanase, N. Thomas, N. Amato, and L. Rauchwerger. STAPL: An adaptive, generic parallel programming library for C++. In *Proc. of the 14th International Workshop on Languages and Compilers for Parallel Computing (LCPC)*, Cumberland Falls, Kentucky, Aug 2001.

[6] M. Apaydin, D. Brutlag, C. Guestrin, D. Hsu, and J.-C. Latombe. Stochastic roadmap simulation: An efficient representation and algorithm for analyzing molecular motion. In *Proc. Int. Conf. Comput. Molecular Biology (RECOMB)*, pages 12–21, 2002.

[7] M. Apaydin, A. Singh, D. Brutlag, and J.-C. Latombe. Capturing molecular energy landscapes with probabilistic conformational roadmaps. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 932–939, 2001.

[8] D. Baker. A surprising simplicity to protein folding. *Nature*, 405:39–42, 2000.

[9] J. Barraquand and J.-C. Latombe. Robot motion planning: A distributed representation approach. *Int. J. Robot. Res.*, 10(6):628–649, 1991.

[10] G. Blelloch. NESL: A Nested Data-Parallel Language. Technical Report CMU-CS-93-129, Carnegie Mellon University, April 1993.

[11] J. Bowie and D. Eisenberg. An evolutionary approach to folding small α-helical proteins that uses sequence information and an empirical guiding fitness function. *Proc. Natl. Acad. Sci. USA*, 91(10):4436–4440, 1994.

[12] J. Bryngelson, J. Onuchic, N. Socci, and P. Wolynes. Funnels, pathways, and the energy landscape of protein folding: A synthesis. *Protein Struct. Funct. Genet*, 21:167–195, 1995.

[13] D. Challou, D. Boley, M. Gini, and V. Kumar. A parallel formulation of informed randomized search for robot motion planning problems. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 709–714, 1995.

[14] D. J. Challou, M. Gini, and V. Kumar. Parallel search algorithms for robot motion planning. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, volume 2, pages 46–51, 1993.

[15] D. Covell. Folding protein α-carbon chains into compact forms by Monte Carlo methods. *Proteins: Struct. Funct. Genet.*, 14(4):409–420, 1992.

[16] V. Daggett and M. Levitt. Realistic simulation of naive-protein dynamics in solution and beyond. *Annu. Rev. Biophys. Biomol. Struct.*, 22:353–380, 1993.

[17] Y. Duan and P. Kollman. Pathways to a protein folding intermediate observed in a 1-microsecond simulation in aqueous solution. *Science*, 282:740–744, 1998.

[18] K. M. Fiebig and K. A. Dill. Protein core assembly processes. *J. Chem. Phys*, 98(4):3475–3487, 1993.

[19] J. Haile. *Molecular Dynamics Simulation: elementary methods*. Wiley, New York, 1992.

[20] L. Kalé, R. Skeel, M. Bhandarkar, R. Brunner, A. Gursoy, N. Krawetz, J. Phillips, A. Shinozaki, K. Varadarajan, and K. Schulten. Namd2: Greater scalability for parallel molecular dynamics. *J. Comp. Phys.*, 151:283–312, 1999.

[21] L. Kavraki, P. Svestka, J. C. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. Automat.*, 12(4):566–580, August 1996.

[22] A. Kolinski and J. Skolnick. Monte Carlo simulations of protein folding. *Proteins Struct. Funct. Genet.*, 18(3):338–352, 1994.

[23] P. Lansbury. Evolution of amyloid: What normal protein folding may tell us about fibrillogenesis and disease. *Proc. Natl. Acad. Sci. USA*, 96(7):3342–3344, 1999.

[24] S. Larson, C. Snow, M. Shirts, and V. Pande. Foldinghome and genomehome: Using distributed computing to tackle previously intractable problems in computational biology. *Computational Genomics*, 2003. To appear.

[25] T. Lazaridis and M. Karplus. Effective energy function for proteins in solution. *Proteins*, 35:133–152, 1999. http://mingus.sci.ccny.cuny.edu/server/.

[26] M. Levitt. Protein folding by restrained energy minimization and molecular dynamics. *J. Mol. Biol.*, 170:723–764, 1983.

[27] M. Levitt and A. Warshel. Computer simulation of protein folding. *Nature*, 253:694–698, 1975.

[28] R. Li and C. Woodward. The hydrogen exchange core and protein folding. *Protein Sci.*, 8(8):1571–1591, 1999.

[29] T. Lozano-Pérez. Spatial planning: A configuration space approach. *IEEE Trans. Comput.*, C-32:108–120, 1983.

[30] T. Lozano-Pérez and P. O'Donnell. Parallel robot motion planning. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 1000–1007, 1991.

[31] V. Muñoz, E. R. Henry, J. Hoferichter, and W. A. Eaton. A statistical mechanical model for β-hairpin kinetics. *Proc. Natl. Acad. Sci. USA*, 95:5872–5879, 1998.

[32] D. Musser, G. Derge, and A. Saini. *STL Tutorial and Reference Guide, Second Edition*. Addison-Wesley, 2001.

[33] L. Rauchwerger, F. Arzu, and K. Ouchi. Standard Templates Adaptive Parallel Library. In *Proc. of the 4th International Workshop on Languages, Compilers and Run-Time Systems for Scalable Computers (LCR)*, Pittsburgh, PA, May 1998.

[34] G. Song, S. Thomas, K. Dill, J. Scholtz, and N. Amato. A path planning-based study of protein folding with a case study of hairpin formation in protein G and L. In *Proc. Pacific Symposium of Biocomputing (PSB)*, pages 240–251, 2003.

[35] M. J. Sternberg. *Protein Structure Prediction*. OIRL Press at Oxford University Press, 1996.

[36] S. Sun. Reduced representation model of protein structure prediction: statistical potential and genetic algorithms. *Protein Sci.*, 2(5):762–785, 1993.

[37] S. Sun, P. D. Thomas, and K. A. Dill. A simple protein folding algorithm using a binary code and secondary structure constraints. *Protein Eng.*, 8(8):769–778, 1995.

[38] O. Tkachyshyn, P. An, G. Tanase, and N. M. Amato. parray as an efficient static parallel container in stapl. Technical Report 02-003, PARASOL Lab, Dept. of Computer Science, Texas A&M University, Aug 2003.

[39] P. Weiner and P. Kollman. Amber: Assisted model building with energy renement, a general program for modeling molecules and their interactions. *J. Comp. Chem.*, 2:287–303, 1981.