Towards petascale *ab initio* protein folding through parallel scripting

Glen Hocky[1], Michael Wilde[2,3*], Joe DeBartolo[4,5], Mihael Hategan[2], Ian Foster[2,3,6], Tobin R. Sosnick[2,4,5*], Karl F. Freed[1,2,7*]
[1]Department of Chemistry, University of Chicago
[2]Computation Institute, University of Chicago & Argonne National Laboratory, USA
[3]Mathematics and Computer Science Division, Argonne National Laboratory, Argonne IL, USA
[4]Department of Biochemistry and Molecular Biology, University of Chicago
[5]Institute for Biophysical Dynamics, University of Chicago
[6]Department of Computer Science, University of Chicago, IL, USA
[7]James Franck Institute, University of Chicago
*wilde@mcs.anl.gov, trsosnic@uchicago.edu, freed@uchicago.edu

Abstract

Petascale computers allow scientists and engineers not only to address old problems better, but also to consider new methods and new problems. We report here on work that both applies new methods and tackles new problems in the area of structural biology. The project combines an efficient protein structure prediction algorithm implemented in the Open Protein System (OOPS) system with the Swift parallel scripting system to enable the rapid and flexible composition of OOPS components into parallel program, and the high-performance execution of these programs on petascale computers. The result is a powerful computational laboratory environment for predicting protein secondary and tertiary structure, for further testing and refining OOPS, and for performing training and scaling tests that enable structure simulation to run on a wide variety of computing architectures with high efficiency. Comparison of before and after experiences within two laboratories at the University of Chicago shows that this use of scripting enables achieving significant improvements in throughput, time-to-solution, and scientific productivity. For example, an undergraduate has recently been able to define and execute new protein folding simulations on thousands of processors. This approach both enables new applications for petascale computers, and provides an avenue for many more researchers to participate in the computational science aspects of structure prediction.

1 INTRODUCTION

To understand a living cell at the microscopic level, we must identify, characterize, and comprehend detailed interactions among sub-components. Decoding the genome has already transformed biology and medicine but is only the starting point of our research and methods, whose long-term goal is to begin with a set of genes identified in a biological process, provide the structure and function of the proteins, and identify their interactions and connections in signaling pathways. Success in this endeavor will lay the foundation for a new generation of therapeutics and drug design.

This research agenda drives our interest in reliable, high-throughput methods for predicting protein structure from sequence and recognize docking partners. The availability of such methods will create a comprehensive resource for understanding these interactions and will eventually replace slower, empirical determinations. To this end, we have developed and continue to use, support, and enhance the Open Protein Simulator (OOPS), a suite of C++ programs and libraries for predicting the structure and interaction of proteins and other large molecules [10]. OOPS has proven successful, through international challenges, such as CASP8, in predicting the structure of moderate size proteins [14].

As OOPS becomes more accurate and efficient, a number of related computational challenges emerge in our desire to tackle proteins of increasing size because current prediction methods have limited accuracy even for proteins on the order of 100 residues when homology-based information is minimal. To predict the structures of larger and multi-domain proteins, statistical sampling becomes a limiting factor, and thus we require significantly more computing resources.

Second, we wish to generate more predictions, of higher, and known, quality, faster, and with less effort required of the users to enable greater focus on the primary intellectual challenges and less on the distracting but necessary efforts involved in performing increasingly parallel, large scale computations. This goal requires both more computing capacity and the ability to specify and execute new OOPS applications rapidly and easily.

Third, progress towards the first two challenges requires the ability to use a wider range of more powerful computers and to reduce barriers to using new computing systems.

In pursuit of these goals, we have sought to create a "protein prediction laboratory" enabling the rapid specification of complex protein prediction applications based on OOPS software. To this end, the cumbersome, inflexible, and manually intensive collection of *ad hoc* shell and Python scripts that had previously been used to drive OOPS have been replaced by the Swift [1] parallel scripting system. Swift provides a high-level syntax that provides for well-structured, abstract, location-independent scripts. The Swift runtime system also automates parallelization, data management, and error recovery, and supports execution on a wide variety of computer systems. This approach allows great flexibility in composing existing programs to address new requirements, to explore algorithmic variations, and to implement entirely new applications, such as new folding and docking algorithms, and replica exchange simulations with multiple order parameters.

The present status report on our progress towards these goals describes the algorithms used by OOPS (Section 2), their computational structure and costs (Section 3), and the parallel scripting approach we employed to extend the power of OOPS (Section 4). Finally, we present in Section 5 results obtained by co-author G. Hocky, an undergraduate student at the University of Chicago, who sought to apply the OOPS Swift-based scripting framework to a range of protein structure prediction problems. These results are anecdotal, but suggestive of the power of the approach.

2 PROTEIN STRUCTURE PREDICTION ALGORITHM

The Open Protein Simulator (OOPS) is a set of open source applications for fast simulation of protein folding, docking and refinement. It uses the C++ protein library PL [3] for representing, moving, and calculating the energy of protein structures, and provides a set of useful analysis tools for evaluating the quality of predicted models. OOPS and its component ItFix algorithm have proven successful at predicting the structure of moderate-sized proteins [13].

The iterative fixing (ItFix) protein structure determination algorithm used within OOPS takes as input a protein sequence, an initial secondary structure, a starting annealing temperature, and other parameters. If successful, it produces a protein structure as output.

Figure 1 shows the basic structure of the ItFix algorithm, which consists of multiple rounds. (The number of rounds is typically limited by a maximum and a convergence test.) At each round, it performs between 100 and 1000 independent, randomly seeded Monte-Carlo-based simulated annealing (MCSA) computations [10]. Then, it gathers statistical data about that round, specifically on the average origins or assignments of the secondary structures at each position in the sequence. Sometimes this analysis involves clustering of structures through various techniques. It then determines whether or not to stop sampling angles from certain secondary structure types at those positions, checks for convergence and, if convergence has not occurred, launches the next round with this information as a new input file. Additionally, at each analysis step, various plots are created from the output data, including average 3D atomic contact maps (and movies), RMSD (3D position accuracy) versus energy plots, and secondary structure prediction accuracy.

The MCSA application called by ItFix consists of a simulated annealing (SA) loop that iterates until it "cools" sufficiently [10]. At each iteration, it rotates the φ/ψ backbone torsional angles in accordance with well-understood physical constraints, seeking moves that produce lower energy and more native-like structures. Thus, each iteration comprises first a move() and then the calculation of the energy of the new configuration, followed by acceptance or rejection of the move based on energy and a temperature-weighted probablity of accepting a higher-energy move.

ItFix and OOPS have unique characteristics that make them well-suited for high-throughput and rapid-response structure prediction and related operations such as simulating the docking configurations of large biomolecules. ItFix incorporates basic chemical principles and mimics a folding pathway to restrict its search space. It employs a highly conditional reduced molecular representation [16], which enables a broader and faster search [10]. In contrast, most other methods include all the atoms and consequently expend much computation time searching through side chain space. Most other algorithms also rely heavily on known structures or fragments (homologies, templates, etc.) [8, 31]. Their success rapidly diminishes as the amount of known information decreases.

3 COMPUTATIONAL CHALLENGES IN PROTEIN STRUCTURE PREDICTION

The ItFix folding algorithm has reduced significantly the time required to predict a protein from sequence, relative to other methods of similar quality. However, in its current implementation, the algorithm still requires ~1000 CPU hours on a modern microprocessor for a medium-sized protein: more than a month.

Fortunately, the hierarchical structure in Figure 1 suggests obvious opportunities for parallel execution. First, the multiple independent invocations of MCSA can be executed in parallel; as each MCSA invocation runs for 0.5 to 3.0 hours, has a small memory footprint (10s of megabytes), and outputs small text files (compressible to < 1MB per MCSA simulation), this strategy is quite straightforward.

```
main(protein, secStr)
  ItFix(protein, nRounds=10, roundSize=300, secStr)

ItFix(protein, nRounds, roundSize, secStr)
   roundNum=1
   while not converged and roundNum < nRounds
      foreach j in 1..roundSize
         models[j], structs[j] = Mcsa(p, secStr)
      newSS = analyze(models, structs)
      converged = checkConvergence(newSecStr, secStr)
      s=newSecStr
      roundNum++

MCSA(protein, secStr)
   initialConf=genRandConf(protein, secStr)
   E = energy(initialConf.model)
   temp, stepsToUpdate, moveSet = GetInitialValues()
   while (not converged)
      position=chooseRandPosition()
      model = move( position, moveSet )
      newE = energy(model)
      model, E = accept_or_reject(E, newE,t)
      if nstep mod steps == 0 then reduce(t)
      nsteps++
   return model, struct
```

Figure 1: The ItFix folding algorithm

Second, it may be possible to exploit parallelism within MCSA. Exploiting for now only the former opportunity, we have produced a code that can compute a single structure in a day on a 150-CPU cluster.

While we plan to consider parallelization of MCSA in the future, that work has not been a priority because our observations of how ItFix and OOPS used in practice within the Freed and Sosnick labs suggest different challenges:

- Researchers often want to invoke ItFix many times at once for different proteins. Thus, even without parallelization of MCSA, we see large runs, involving hundreds of thousands of independent activities, with associated procedural challenges in terms of bookkeeping, error detection, restarting, and so forth. (See Figure X.)
- The ItFix algorithm has several free parameters that are currently trained by chemical intuition and repeated trials. In order to best understand performance, researchers often want to run extensive benchmarks to evaluate algorithmic improvements. Again, the result is large runs involving many invocations of MCSA and other procedures.
- Researchers often make changes to the computational structure—not at the lowest level of the MCSA application, but to things like convergence criteria.
- Researchers also experiment with new applications of the OOPS framework, for example to crystal structure refinement and studies of biomolecular interactions. These applications do

not change the basic MCSA building block, but do again involve the specification of new high-level structure.

- Because computing demands always seem to exceed available resources, researchers frequently seek to make use of multiple computers, including local clusters and computers at NSF and DOE centers.

These observations on how ItFix and OOPS are used suggest a need for mechanisms that allow for concise, readable specification of high-level structure that exposes opportunities for parallel execution; the robust management of large number of tasks; and the convenient dispatch of computation to multiple parallel computers, both local and remote.

These considerations motivated our exploration of the use of the Swift parallel scripting system within OOPS.

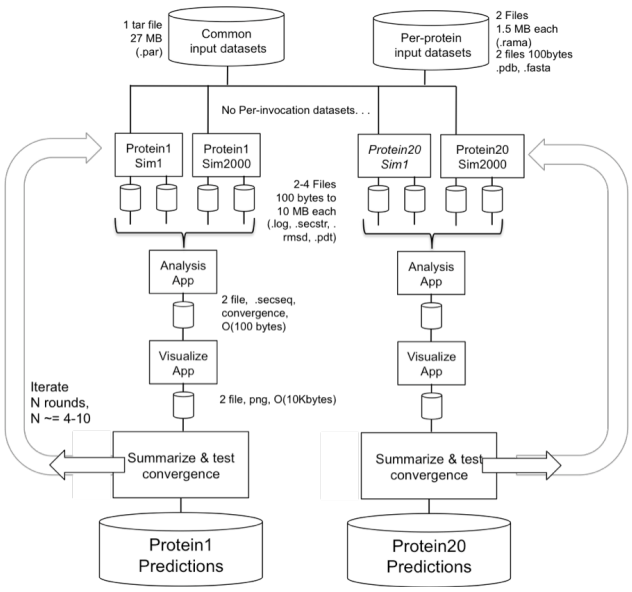

Figure 2: OOPS ItFix data flow diagram, showing algorithm applied to multiple proteins concurrently, in multiple rounds.

4 PARALLEL PREDICTION SCRIPTING

Swift [29] comprises:

1. a high-level, functional scripting language ("SwiftScript"), designed for expressing computations that invoke executable programs, with a dataflow model used to ensure that program invocations ("tasks") are executed only when their input data is available;
2. a data model that allows for the mapping of file system structures (individual files, directories, etc.) into Swift language variables; and
3. a runtime system to manage the scheduling of tasks for execution, the dispatch of executable tasks to parallel computers, and the movement of data consumed and produced by those tasks.

```
1.  app (PDT structure) predict(Fasta protein) {
2.      predict_structure @protein @structure;
3.  }
4.
5.  foreach pfile, i in @arg("proteins") {
6.     Fasta protein <pfile>;
7.     PDT structure[];
8.     structure[i] = predict(protein);
9.  }
```

Figure 3: A simple Swift example

Figure 2 illustrates some basic Swift constructs. In brief:

- (L1-3) Defines an interface to an application program, **`predict_structure`**. This interface maps from typed Swift variables to command-line program syntax. It expects a protein sequence specified in FASTA format and returns a structure prediction in the form of a "trajectory" file in our "PDT" format.
- (L5-9) Invokes the procedure **`predict`** (and thus the **`predict_structure`** program) in parallel, for each file listed in the command line argument "proteins."

Swift's dataflow model means that the multiple invocations of **`predict`** can run concurrently, as none is dependent on data produced by another. Swift's runtime system handles the dispatch of individual predict calls to an available computer, and the movement of the associated data to and from those computers. Thus, we are able in just eight lines to describe a potentially large amount of computing.

The Swift runtime system can deal with computations that involve many concurrent activities. By using two-level scheduling methods, as implemented for example in Falkon [24, 25], it has executed computations involving hundreds of thousands of tasks on supercomputers with tens of thousands of processors. (These methods first deploy task executors onto nodes and then stream tasks to those executors.) The Swift runtime system can also dispatch tasks to multiple computers, using Globus mechanisms [17] to overcome inevitable heterogeneities in authentication, job submission, and data movement methods. Swift implements various heuristics to decide where to send which task, and when.

When we first discussed combining OOPS with Swift, we sketched out simple pseudo-code examples, similar to Figure 1, of how we would use Swift programs to combine OOPS functions into high-performance applications. Our original conception has stood the test of time. While the integration of the OOPS framework with Swift demanded some extra initial development time, this investment has paid off. We have more concise and manageable programs; can run complex computations more easily and reliably; and are able to run our programs across many sites without modification.

To illustrate how we use Swift in OOPS, we present a simplified version of the basic ItFix program structure. The complete currently executing Swift script is available online [2].

We leverage Swift data typing and mapping [20] to abstract input and outputs, group related items in structures, detect type errors, and map the simple logical structure to the specific data layout that we want to maintain in our archival storage repository. We first declare some useful atomic types to be simple files: for example, Fasta for the sequence being folded, and PDB, the known 3D structure, when available, for accuracy comparison. (Other such simple types are elided). Then we define some compound types, which as in other languages, are used to organize multiple related values.

```
1.  type Fasta;
2.  type PDB;
3.  ...
4.
5.  type MCSAIn {
6.    Fasta fasta;
7.    PDB pdb;
8.    SecSeq secseq;
9.  }
10.
11. type MCSAOut {
12.   OOPSSecStr SecStr;
13.   OOPSLog log;
14.   OOPSEnergy Energy;
15.   OOPSpdt pdt;
16.   OOPSrmsd rmsd;
17.   OOPSLibrary Library;
18. }
```

We use **app** procedures to define Swift interfaces to application codes. For example, the Swift procedure **mcsa** defines an interface to the **oops_mcsa** program (L19). Note how the procedure extracts components from the Swift procedure arguments and uses them to construct the arguments to **oops_mcsa** (L20).

```
19. app (MCSAOut out) mcsa (MCSAIn i, SecSeq secseq,
                        int jobnum, string cfgParams[])
    {
20.    oops_mcsa @i.fasta @secseq @i.pdb @out.pdt
           @out.rmsd jobnum cfgParams stdout=@out.log;
21. }
```

We often also find it useful to define Swift interfaces to small utility functions. For example, in the following we use the Unix **sed** program to replace the values of science parameters in OOPS run configuration files. Any of the approximately 50 scientific parameters in about 8 configuration files can be dynamically set, and used in a parameter sweep, in this manner. In L22-24 below, we set parameters values that control the simulated annealing temperature ranges and descent rates.

```
22. app (file oParams) setTemps (file inParams, string
                                 start, string update)
    {
23.    sed "-e" @strcat("s/@DIT@/",start,"/") "-e"
           @strcat("s/@TUI@/",update,"/")
           @inParams stdout=@oParams;
24. }
```

Next, we define our parallel application logic. First, we specify how a single ItFix round is performed, via multiple concurrent calls to **mcsa**. The procedure **singleRound** (L25) sets various science configuration parameters (L28-29) and then uses a **foreach** statement (L30) to make the multiple calls to **mcsa**, accumulating the outputs in the array **out**.

```
25. (MCSAOut out[]) singleRound
    (string protein, MCSAIn mcsaIn, SecSeq secSeq,
    int round, int nsim, string startTemp,
    string tempUpdate)
26. {
      file inParams <@arg("params")>;
27.   file editedParams =
         setTemps(inParams, startTemp, tempUpdate);
28.   string config [] = readData(editedParams);
29.
30.   foreach sim in [0 : (nsim-1)] {
31.     out[sim] =
            mcsa(mcsaIn, secseq, sim, config);
32.    }
33. }
34.
```

The procedure **ItFix** (L35) implements the ItFix algorithm, calling **singleRound** repeatedly (and serially) (L48) until either convergence is detected (L51-52) or the specified rounds limit is reached (L52).

```
35. ItFix(string protein, int nsim, int maxrounds,
          string startTemp, string tempUpdate)
36. {
      OOPSIn oopsin <ext; exec="OOPSIn.map.sh",
                          i="input", p=protein>;
37.
38.   string outdir = @arg("outdir");
39.   OOPSOut result[][] <ext;
              exec="SecSamplerOutAll.map.sh",
40.           d=outdir, p=protein, r=maxrounds,
              s=nsim, t=startTemp, u=tempUpdate>;
41.   SecSeq secseq[] <simple_mapper; prefix =
         @strcat(outdir, "/", protein, "/", protein,
42.              ".ST", st, ".TU", tu, "."),
43.      suffix=".secseq">;
44.   boolean converged[];
45.   external done[];
46.   secseq[0] = cpSecSeq(oopsin.secseq);
47.
48.   iterate i {
49.     (done[i], result[i]) =
          singleRound(protein, oopsin, secseq[i],
50.                   i, nsim, startTemp, tempUpdate);
51.     (converged[i], secseq[i+1]) =
          analyzeRoundDir(protein, i, secseq[i],
          done[i]);
52.   } until ( converged[i] || (i==(maxrounds-1)) );
53. }
```

We can now provide the main program, in which we call may ItFix directly, to predict the structure of a single protein, or alternatively build up more complex programs. For example, the following program runs each of a set of protein sequences (from file **plist**), in up to **maxrounds** rounds:

```
54. main_loop()
55. {
56.   int nsim = @toint(@arg("nsim"), 3);
57.   int maxrounds = @toint(@arg("maxrounds", "3"));
58.   string protein[] = readData(@arg("plist"));
59.   foreach prot in protein {
60.     ItFix(prot, nsim, maxrounds,"","");
61.   }
62. }
```

Thus, the simple code fragment in lines 54-62 above, given 10 proteins, nsim=1000, would, in each round of up to 3 rounds of prediction, execute 10 x 1000 = 10,000 simulations. The actual degree of parallelism is controlled by swift runtime settings and by the availability of processor resources.

5 EXPERIENCE ON LARGE SYSTEMS

As the main work of our group is algorithm and method development, we are continually testing and evaluating the accuracy, spatial, and time performance of new codes. The OOPS code base is constantly evolving. Prior to deployment of our parallel scripting methods, it was virtually impossible to continually test evolving changes at scale. The methods described here make a new approach possible.

We are already using this framework to test some improvements to the ItFix algorithm. In this section we show some examples of what you can do in short main programs, once the core library routines above have been created and validated.

5.1 Usage rates

In the first two weeks of April 2009, just shortly after the ItFix Swift script was developed, the system has seen impressive use in pursuit of scientific inquiries by author Hocky:

ALCF Intrepid Blue Gene/P:

67178 jobs, 208,763 CPU hours

TeraGrid:

22495 jobs, 2397 CPU hours

Ranger:

17488 jobs, 1425 CPU hours

Over 100 GB of compressed science results data was produced from the Blue Gene runs alone.

5.2 Parameter sweeps

Given the library of Swift procedures defined above, the programmer can use flexible scripts to leverage many processors with relative ease, as in the following code.

```
int nsim      = @toint(@arg("nsim"), 3);
int maxrounds = @toint(@arg("maxrounds", "3"));
string protein[]    = readData(@arg("plist"));
string startT[] = readData(@arg("startT"));
string tUpdate[] = readData(@arg("tUpdate"));

foreach prot in protein {
  foreach sT in startT {
    foreach tUp in tUpdate {
      ItFix(prot, nsim, maxrounds, sT, tUp);
    }
  }
}
```

This simple code fragment, given 10 proteins, nsim=1000, two starting temperatures and 5 update intervals, would, in each round of up to 3 rounds of prediction, execute 10 x 1000 x 2 x 5 = 100,000 simulations. On highly parallel systems such as the Argonne Intrepid BGP, this simple code fragment can fully utilize a substantial portion of the machine's 163,840 processor cores. Similar code with a slightly more general parameterization of ItFix can sweep across any combination of settable parameters that govern the OOPS MCSA algorithm.

5.3 Data analysis and visualization

We have integrated a range of visualization tools (e.g., PyMol for protein visualization; scatter plots of protein energy level vs. the root-mean-square distance (RMSD) of backbone atoms of the predicted structure to the known structure) into the framework via simple analysis scripts. These visualizations are a primary tool used by our labs to assess prediction quality. Analysis in the current script is accomplished by collectively summarizing round results from log files, generating diagnostic 2D secondary structure predictions, and calculating the lowest RMSD and predicted 3D models. Quantitative molecular accuracy results and molecular visualization are also produced in a web-based format (Figure 7).

5.4 OOPS Experiments conducted in Swift

As mentioned, the ItFix algorithm has several free parameters. Each individual folding simulation is a Monte Carlo Simulated Annealing (MCSA) procedure, meaning that the simulation is started at a Starting Temperature (ST) and after a Temperature Update Interval (TUI) the temperature is decreased based on a temperature scheduling algorithm [10]. The idea is to have a simulation that drives a molecule its lowest energy state, which in the case of proteins we call the native state. This occurs because in the MCSA loop, a structure is accepted only if it has a lower energy then the previous, or if it has a higher energy, on with a conditional probability decreasing with temperature.

We use a statistical scoring function as our measure of energy, and because our temperature units are arbitrary it is difficult to determine what values of ST and TUI will give the desired results. While the ItFix procedure as implemented in Debartolo et al. [13] is highly successful, the ST and TUI parameter space were not explored extensively. We wanted to know for that particular implementation, and for future implementations of ItFix, whether other combinations of these parameters can give comparable or better results while using less computer time.

Since we already had a flexible Swift framework implemented for running OOPS across multiple sites, it was simple to implement a parameter sweep workflow that explores this space and which leverages HPC resources to do so. This parameter sweep is essentially the program provided in Section 5.1. Figure 5 describes our use of HPC and results.

Protein	Length	Class	ST	TUI	Lowest RMSD (Å)	DeBartolo RMSD (Å)	Protein	Length	Class	ST	TUI	Lowest RMSD (Å)
T1af7	69	α	15	25	3.77	2.5	T1dcj	72	α/β	15	25	8.75
				50	3.60						50	9.11
				100	3.77						100	7.22
			25	25	3.20					25	25	8.34
				50	3.78						50	7.69
				100	3.01						100	8.94
T1r69	61	α	15	25	3.20	2.4	T1ubq	73	α/β	15	25	6.68
				50	4.09						50	7.05
				100	3.87						100	6.00
			25	25	3.76					25	25	6.88
				50	2.94						50	8.29
				100	3.87						100	8.01

Figure 4 – The outputs of an initial test run on Intrepid are shown here for four representative proteins. Note that the protein's native secondary structure was used as input, so a direct comparison to DeBartolo et al. is only relevant for *α*-proteins, where ItFix effectively converges to the native secondary structure.

5.4.1 Initial test

In order to evaluate the performance of the ItFix algorithm, we chose proteins whose structures are well determined experimentally, and give as input to the ItFix algorithm the native secondary structure in addition to the amino acid sequence. In the method of DeBartolo et al., to generate predictions for publication, the ST was 100 and the TUI was 1000 and 2000 simulations were run for each protein. In our initial test, we ran short simulations on Argonne's BG/P (Intrepid). For four proteins, we ran 150 simulations each of ST of 15 and 25 and TUI of 25, 50, and 100 (see **Figure 4**).

The results presented immediate information for further investigation. The four proteins we picked were representative, two contained only the α-helical secondary structure unit, and two contained both α and β motifs. For the α/β proteins the results were predictably bad. We knew from experience that a large amount of sampling must be done to get properly aligned β structures. However, the results for α proteins were surprising. For all TUI ad ST combinations and in just 150 simulations, *the results were nearly comparable to those published by DeBartolo et al., while using two orders of magnitude less simulation*. The obvious

conclusion was that once a protein is assigned as α by the ItFix algorithm, we should consider running many short simulations rather than fewer long simulations. Two questions then remained: (a) can we use this method to generate predictions of higher accuracy than DeBartolo et al. with this method, and (b) what should we do about α/β and β proteins?

5.4.2 Investigation of α proteins

Because we were running short simulations, this investigation was particularly fruitful. We used a mixture of Teragrid sites for these tests (Abe, QueenBee, and Ranger), where simulations all took 2-10 minutes. The results in this test were strong. Running approximately 5000 simulations for each of three alpha proteins resulted in results comparable or better than those of DeBartolo et al. Additionally, for each protein the CPU time utilized was approximately 500 CPU-hours while that used in the ItFix protocol would take more than 2000 CPU-hours depending on system load, etc.

Protein	Length	ST	TUI	Lowest RMSD (Å)	DeBartolo RMSD (Å)
T1af7	69	25	100	2.07	2.5
T1b72	50	25	100	1.41	1.6
T1r69	61	25	100	2.11	2.4

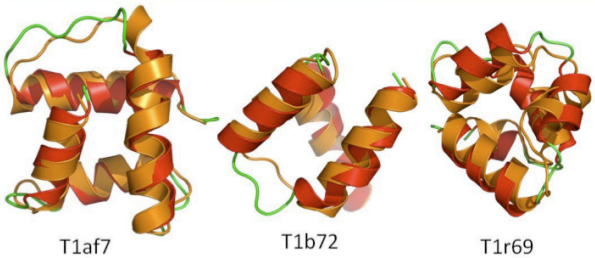

Figure 5 -- Table presenting results of running > 5000 simulations on alpha proteins using TeraGrid resources. The structures generated are better than those of DeBartolo et al. **[13]**, and required significantly less computation. The lower images are the predicted structures (with helices in red and coil in green) overlaid with the native structures (colored orange).

5.4.3 Investigation of β proteins

Parameter sweep simulations for β proteins demonstrated the anecdotal evidence described above; it is necessary to do a large amount of in-simulation sampling in addition to many parallel MCSA runs to generate a good ensemble of structure for β or α/β targets. Since this was the case, we then decided to investigate the performance of Intrepid with our simulation framework for use in future folding investigations where large amounts of sampling are necessary.

5.4.4 Replicate/compare DeBartolo et al. runs

In previous sections, we used our scripting framework to investigate properties of ItFix. Another thing we wish to learn is what resources will be useful for which kinds of future applications. One thing we would like to know is whether a resource such as a BG/P with many low-power processors can be useful for our types of studies. In Figure 6 we show that an ItFix investigation can be successfully and stably executed with Swift/Falkon. Though the runtimes are longer than would be expected on stock processors, the availability of thousands of processors and the stability of our system shows that use of Intrepid can be fruitful in future investigations. After this simulation we also successfully ran simulations with the exact protocol of DeBartolo et al. for T1mky and T1tif, results of which can be seen on our project web site [2].

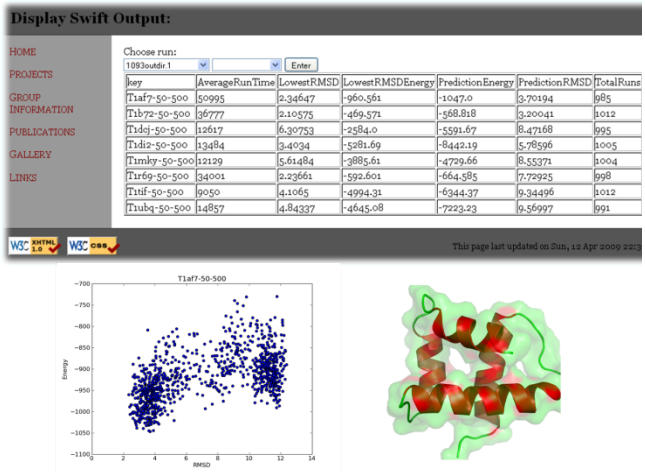

Figure 6 – Results of running eight proteins on 2 racks (8192 CPUS) on Argonne's BG/P, Intrepid. Below are results from this investigation for T1af7. On the left is a scatter plot showing the correlation between our statistical energy potential and accuracy of the protein structures for the 985 simulations that ran to completion. On the right is an image showing the lowest RMSD structure. This table, plot and image were all automatically generated by our scripting mechanism, and the table is presented by a simple CGI script at our web site [2].

5.4.5 Automatic Timing Runs

To effectively utilize computational resources for Monte Carlo simulations, it is important to know how your algorithm scales with the amount of sampling that you wish to have done. With this framework, we can easily run loops over various parameters and test the scaling of our algorithm. We performed a simple test on the TeraGrid site Ranger, using three different ST values (25,50,100) and 4 different TUIs (50,100,200,500). A sample output can be seen in Figure 7.

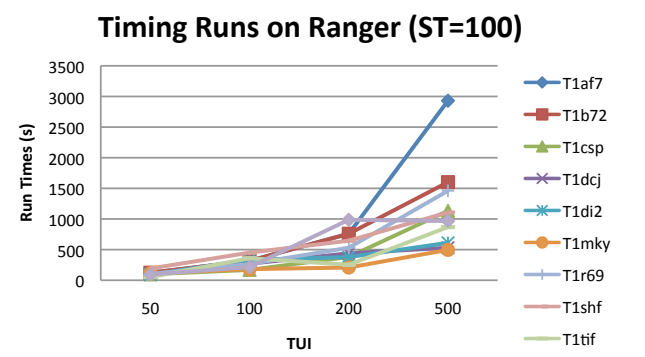

Figure 7 – Sample result of a timing run on Ranger for varying values of Temperature Update Interval with Starting Temperature of 100.

6 RELATED WORK

We discuss related work in both protein structure prediction and computing.

6.1 Structure prediction approach

Most folding algorithms rely heavily on known structures or fragments (homology- or template-based) and can be extremely successful [8, 31]. Their success, however, rapidly decreases as the amount of known information decreases [26]. Other methods take a physics-based approach [22], but are limited in their ability to predict large targets on short time scales.

Uniquely among known approaches to structure prediction, OOPS operates with minimal use of information derived from sequence similarity to proteins in the Protein Data Bank (PDB) [7]. OOPS derives its speed and accuracy from the use of a "Cβ" model [10], an accurate statistical potential [16], and a search strategy involving iterative fixing of structure in multiple "rounds" of folding [13]. Since OOPS uses minimal homology information and a reduced representation, its success depends crucially on describing the protein physics correctly. Great effort has been devoted to the energy function, e.g., interactions are conditional on backbone geometry and the relative orientation of side chains. In 2007, its accuracy exceeded available all-atom potentials [16] by utilizing secondary structure dependence, and it has been significantly improved since then by including orientation-dependence [13]. Its homology-free secondary and tertiary structure predictions for small proteins rival or exceed homology-based methods with (expensive) explicit side chains, engendering optimism for continued progress. It also employs sequence homology for additional accuracy when appropriate.

Our algorithm can predict the structures of two sets of proteins with comparable accuracy for α, α/β, and β proteins (DeBartolo et. al , [13] Table 2). These predictions are comparable in accuracy to the successful Rosetta fragment-based insertion algorithm, described in the papers from which the test sets are obtained [8, 19].

6.2 Computing approach

Our computing approach builds on two related layers: the Swift parallel scripting system, and two-level scheduling, as implemented in Falkon and via the Swift "coaster" mechanism.

Other approaches to high-level specification of loosely coupled scientific computations include MapReduce, DAGMan, Pegasus, BPEL,Taverna. Triana, Kepler, and Karajan.

MapReduce [12] supports the processing of key-value based data, using the Google File System. Swift targets various scientific applications that process heterogeneous data formats, and can schedule computations in a location-independent way.

Pegasus [15] and DAGMan [4] can also schedule large scale computations in Grid environments. DAGMan provides a workflow engine that manages Condor jobs organized as directed acyclic graphs (DAGs) in which each edge corresponds to an explicit task precedence. It has no knowledge of data flow, and in distributed environments works best with a higher-level, data-cognizant layer. DAGMan also lacks dynamic features such as iteration (which can result in large DAGs) and conditional execution. Pegasus is primarily a set of DAG transformers that can translate a workflow graph into a location-specific DAGMan input file; prune tasks for files that exist; select sites for jobs; and cluster jobs based on various criteria. A weakness is that planners must operate on an entire workflow statically, and execution sites cannot be changed after Pegasus processes a workflow, which can be long before a job runs, a strategy that may not work well in dynamic environments.

BPEL [1] has primarily been applied in service composition and orchestration. A lack of support for iteration means that programs can be larger, although the problem is being addressed in its latest 2.0 specification. In addition, the complex XML specification is cumbersome to write compared with our compact scripting language.

Taverna [21], Triana [27], and Kepler [5] have also been applied in science problems. However, they do not abstract dataset types or provide location transparency. Data movement and Grid job submission all need to be explicitly specified and organized. Their support for multi-site Grid execution is also of limited scale.

As discussed earlier, Swift integrates Karajan [18]. Karajan provides the libraries and primitives for job scheduling, data transfer, and Grid job submission; Swift adds support for high-level abstract specification of large parallel computations, data abstraction, and workflow restart, and also (via Falkon) fast, reliable execution over multiple Grid sites.

Work related to Swift's use of Falkon and coasters for lightweight scheduling includes IBM's Kittyhawk project [6], Cope et al.'s work [11] on integrating lightweight scheduling in the Cobalt scheduling system, using the HTC-mode support in Cobalt. We have compared the performance of Falkon on the Blue Gene/P with those of Cope at al. and Peters et al. [23], and found at least one order of magnitude better performance, and several orders of magnitude better scalability. This improved performance and scalability of the middleware can translate into direct improvements in scalability and performance for applications, with finer grained task parallelism and reduced end-to-end application execution times.

7 FUTURE WORK

The work described here has come together over a fairly short period in Feb-April 2009. Its success has enabled us to chart the following enhancements, which our experience to date suggests are readily achievable.

While it was straightforward to code ItFix in Swift, the language and its runtime semantics are still young and evolving. We were eager to learn more about the language's completeness and usability through the experience of the kinds of real production usage as this project. Three aspects of language enhancement seem desirable from our OOPS scripting experience: 1) the need for polymorphism, so a superset data structure can receive the results of several similar but non-identical application program signatures; 2) the desirability of providing global variables so that, for example, all functions can be aware of application command line variables; and 3) an alternative to the Swift construct called "external" variables, used to circumvent the systems inability to pass a very long list of data objects as a command line argument. All of these are now being considered for near-term enhancements to the language.

Scaling through hybrid parallelization. To improve the scaling of OOPS on systems such as the BG/P, with massive numbers of low-speed processors, we will turn OOPS into a hybrid HPC/MTC application—what we may term "MPTC," for many parallel-task computing—by parallelizing the MCSA energy-computing function, in which profiling has shown that over 85% of the time in OOPS is spent. We have designed a data structuring approach that will make this straightforward, opening the door to much greater scaling by using up to 1024 cores for a single MCSA prediction operation.

We can thus estimate the speeds we expect to achieve on our target petascale platforms. Folding 10 proteins with a fold size of 10 rounds and a round size of 1,000 MCSA simulations can utilize 100,000 CPUs working in parallel with no data dependencies and hence near-perfect scaling. If we devote between 4 and 16 cores to each Mcsa() function for the parallel computation of the energy of each configuration, we can effectively utilize 40,000 to 160,000 compute cores for this task, or 4,000 to 16,000 cores per protein. Realistically, even on petascale systems with 50K to 300K cores, most user jobs will run with allocations far less than the full system. Thus, this scale fits well for today's usage, and can expand to greater utilization, even for single proteins, as the parallelization of the energy computation increases.

Many-task data management. Loosely coupled parallel scripting, while productive for the developer, imposes a high performance burden on large scale systems. We address this issue with a collective I/O model for file-based many-task computing [28] that we have prototyped on the BG/P and which enables efficient distribution of input data files to computing nodes and gathering of output results from them. This approach broadcasts common input data, and uses efficient scatter/gather and caching techniques for input and output.

Comprehensive user environment. An OOPS "run configurator" mechanism packaged for use both from a web-form-based interface as well as via a simple textual command specification will enable users to specify OOPS runs with no programming. The web interface will be runable locally by any user or community as a service of the OOPS workflow framework.

The collaboration environment will leverage the Computation Institute's Petascale Active Data Server (PADS): a 0.5PB storage system integrated with a 384-node cluster, with another 0.3PB of storage and ample RAM on the cluster nodes (NSF grant OCI-0821678). This facility will be ideal for "stage-2" analysis (where stage-1 analysis is done on the target petascale systems themselves as part of the OOPS workflow, as described above).

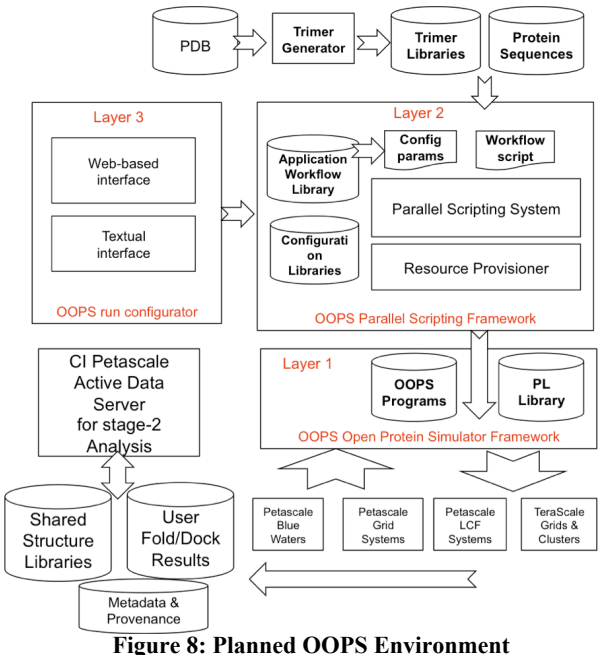

Figure 8: Planned OOPS Environment

Tools such as R, Octave, and MatLab can be readily integrated into analysis scripts (as many Swift users do today). Such analysis scripts can utilize the same parallel scripting language as the OOPS run-time framework, and can run both on the target petascale systems as well as the backend "stage-2 analysis" environments such as PADS, clusters, and workstations.

8 CONCLUSION

We have described the recent, rapid success in recoding an ad-hoc implementation of protein structure prediction by "iterative fixing" and simulated annealing in the Swift parallel scripting language, and report on the progress, benfits, and remaining work needed to make this approach an even more highly-productive example of utilizing petascale systems to achieve greater scientific insights into important aspects of the structural and behavioral properties of large biomolecules.

We have identified remaining deficiencies in this approach and presented a plan for future work that addresses them.

In general, we believe our work shows, in part, the unsurprising conclusion that easier access to a greater level of computing resources means a larger lab in which to test more hypotheses, in less time, with less effort, and thus few distractions for scientists seeking to advance their science rather than to address the complexities of computing at this scale.

We believe our work demonstrates that once a basic set of procedures have been created, the Swift approach to parallel scripting can be productive; what we find most exciting is that short, compact concise scripts, which clearly show the science logic, can be automatically executed across diverse resource types, and can leverage large computational resources. Thus, we run much larger problems, and explore a large scientific space. It allows us to ask questions we could not ask before.

Another advantages of our script-based approach is that we have the future benefit of automated data provenance tracking [9, 30] within the workflow execution engine, as well as the ability to leverage multiple petascale execution resources within a single computation.

From our experience with the system to date, we believe that the new capability will accelerate the rate of discovery in the Freed and Sosnick labs, by increasing the rate at which we can improve the speed and accuracy of OOPS. A core science process in these labs is the enhancement in terms of predictive accuracy, speed, and functional capability of the Open Protein Simulator. Enabling many more lab members to perform more simulations, with larger Monte Carlo sample sets, and to easily test and compare the performance of the system across a range of parameter values, has proven its value already in terms of new insights into the behavior of our algorithms and the degree of simulation needed to converge on accurate predictions.

ACKNOWLEGEMENTS

This work was supported in part by the National Science Foundation under Grant OCI-0721939, by NASA Ames Research Center GSRP Grant Number NNA06CB89H; by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Dept. of Energy, under Contract DE-AC02-06CH11357; by NIH research (T.R.S. and K.F. F.) and training grants, and by the National Science Foundation through TeraGrid resources provided by NCSA, by the ALCF, and resources of the

Open Science Grid. The authors would like to thank: The Swift Group, the Computation Institute, the ZeptoOS group and the Argonne Leadership Computing Facility

REFERENCES

1. Business Process Execution Language for Web Services, Version 1.0, http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/, 2002.
2. Open Protein Simulator computation tracking web site. [cited 2009 April 13]; Available from: http://freedgroup.uchicago.edu/oops.html.
3. PL protein library. [cited 2009 April 13]; Available from: http://protlib.uchicago.edu.
4. The Condor DAGMan (Directed Acyclic Graph Manager), 2007.
5. Altintas, I., Berkley, C., Jaeger, E., Jones, M., Ludäscher, B. and Mock, S., Kepler: An Extensible System for Design and Execution of Scientific Workflows. in *16th Intl. Conference on Scientific and Statistical Database Management*, (2004).
6. Appavoo, J., Uhlig, V. and Waterland, A. Project Kittyhawk: Building a Global-Scale Computer -- Blue Gene/P as a Generic Computing Platform. *ACM Sigops Operating System Review*.
7. Berman, H.M., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T.N., Weissig, H., Shindyalov, I.N. and Bourne, P.E. The Protein Data Bank. *Nucleic Acids Res*, *28* (1). 235-242.
8. Bradley, P., Misura, K.M. and Baker, D. Toward high-resolution de novo structure prediction for small proteins. *Science*, *309* (5742). 1868-1871.
9. Clifford, B., Foster, I., Voeckler, J., Wilde, M. and Zhao, Y. Tracking Provenance in a Virtual Data Grid. *Journal of Concurrency and Computation, Practice and Experience*.
10. Colubri, A., Jha, A.K., Shen, M.Y., Sali, A., Berry, R.S., Sosnick, T.R. and Freed, K.F. Minimalist representations and the importance of nearest neighbor effects in protein folding simulations. *J Mol Biol*, *363* (4). 835-857.
11. Cope, J., Oberg, M., Tufo, H.M., Voran, T. and Woitaszek, M. High Throughput Grid Computing with an IBM Blue Gene/L *IEEE International Conference on Cluster Computing*, Austin, TX, 2007.
12. Dean, J. and Ghemawat, S. MapReduce: Simplified Data Processing on Large Clusters *6th Symposium on Operating System Design and Implementation,*, San Francisco, CA, 2004.
13. DeBartolo, J., Colubri, A., Jha, A.K., Fitzgerald, J.E., Freed, K.F. and Sosnick, T.R. Mimicking the folding pathway to improve homology-free protein structure prediction. *Proc Natl Acad Sci U S A*, *106* (10). 3734-3739.
14. DeBartolo, J., Hocky, G., Zhou, F., Peng, J., Augustyn, A., Adhikari, A., Xu, J., Freed, K.F. and Sosnick, T.R. Structure prediction combining the template-based RAPTOR algorithm with the ItFix ab initio method *CASP8 conference*, http://predictioncenter.org/casp8/, 2008.
15. Deelman, E., Singh, G., Su, M.-H., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Vahi, K., Berriman, G.B., Good, J., Laity, A., Jacob, J.C. and Katz, D.S. Pegasus: A Framework for Mapping Complex Scientific Workflows onto Distributed Systems. *Scientific Programming*, *13* (3). 219-237.
16. Fitzgerald, J.E., Jha, A.K., Colubri, A., Sosnick, T.R. and Freed, K.F. Reduced C(beta) statistical potentials can outperform all-atom potentials in decoy identification. *Protein Sci*, *16* (10). 2123-2139.
17. Foster, I. Globus Toolkit Version 4: Software for Service-Oriented Systems. *Journal of Computational Science and Technology*, *21* (4). 523-530.
18. Laszewski, G.v., Hategan, M. and Kodeboyina, D. Java CoG Kit Workflow. in Taylor, I.J., Deelman, E., Gannon, D.B. and Shields, M. eds. *Workflows for Science*, 2007, 340-356.
19. Meiler, J. and Baker, D. Coupled prediction of protein secondary and tertiary structure. *Proc Natl Acad Sci U S A*, *100* (21). 12105-12110.
20. Moreau, L., Zhao, Y., Foster, I., Voeckler, J. and Wilde, M. XDTM: XML Data Type and Mapping for Specifying Datasets *European Grid Conference*, 2005.
21. Oinn, T., Addis, M., Ferris, J., Marvin, D., Senger, M., Greenwood, M., Carver, T., Glover, K., Pocock, M.R., Wipat, A. and Li, P. Taverna: A tool for the composition and enactment of bioinformatics workflows *Bioinformatics Journal*, *20* (17). 3045-3054.
22. Ozkan, S.B., Wu, G.A., Chodera, J.D. and Dill, K.A. Protein folding by zipping and assembly. *Proc Natl Acad Sci U S A*, *104* (29). 11987-11992.
23. Peters, A., King, A., Budnik, T., McCarthy, P., Michaud, P., Mundy, M., Sexton, J. and Stewart, G. Asynchronous Task Dispatch for High Throughput Computing for the eServer IBM Blue Gene® Supercomputer *International Parallel and Distributed Processing Symposium*, 2008.
24. Raicu, I., Zhang, Z., Wilde, M., Foster, I., Beckman, P., Iskra, K. and Clifford, B. Toward loosely coupled programming on petascale systems *2008 ACM/IEEE Conference on Supercomputing*, 2008.
25. Raicu, I., Zhao, Y., Dumitrescu, C., Foster, I. and Wilde, M. Falkon: a Fast and Light-weight tasK executiON framework for Grid Environments *SC'2007*, 2007.
26. Service, R.F. Problem solved* (*sort of). *Science*, *321* (5890). 784-786.
27. Taylor, I., Shields, M., Wang, I. and Harrison, A. Visual Grid Workflow in Triana. *Journal of Grid Computing,*, *3* (3-4). 153-169.
28. Zhang, Z., Espinosa, A., Iskra, K., Raicu, I., Foster, I. and Wilde, M. Design and Evaluation of a Collective IO Model for Loosely Coupled Petascale Programming *IEEE Many-Task Computing on Grids and Supercomputers*, Austin, TX, 2008.
29. Zhao, Y., Hategan, M., Clifford, B., Foster, I., von Laszewski, G., Nefedova, V., Raicu, I., Stef-Praun, T. and Wilde, M. Swift: Fast, Reliable, Loosely Coupled Parallel Computation *1st IEEE International Workshop on Scientific Workflows*, 2007, 199-206.
30. Zhao, Y., Wilde, M. and Foster, I. Applying the Virtual Data Provenance Model *International Provenance and Annotation Workshop*, 148-161, Chicago, IL, USA, 2006.
31. Zhou, H. and Skolnick, J. Protein structure prediction by pro-Sp3-TASSER. *Biophys J*, *96* (6). 2119-2127.