A review of recent advances in *ab initio* protein folding by the Folding@home project

William Ito

Abstract

The Folding@home project harnesses a volunteer distributed computing network to perform *ab initio* molecular simulations of protein folding. Thanks to engineering innovations like a Graphical Processing Unit (GPU) client for running simulations, Folding@home is able to harness over 6.1 petaFLOPS of processing power, allowing it to simulate longer and more complex protein folding mechanisms than ever before. This paper will review the background science underlying the Folding@home project with a focus on the GPU client and its use in folding NTL9, the slowest-folding protein ever simulated *ab initio* with fidelity to experimental results.

1. Introduction

Protein folding is an important process in which a polypeptide chain spontaneously folds into its functional three-dimensional protein structure. Understanding this process is fundamental to our understanding of biology - many diseases including Alzheimer's, bovine spongiform encephalopathy (mad cow disease), and Parkinson's are associated with the aggregation of non-functional proteins due to misfolding [1][2]. Despite this importance, our understanding of the exact pathway by which proteins fold remains limited. We are only able to study snap-shots of protein structures by X-ray crystallography and NMR. Computer simulations of protein folding and dynamics pose a compelling way to study protein folding in action, but have been traditionally challenging due to the physical complexity and timescales involved in this process.

A modern single-core processor can perform around 5 nanoseconds of simulation per day for a small (544 atom) protein, dropping to picoseconds per day for a larger (~5000 atom) protein [3]. However, most protein dynamics of interest occur over milliseconds to seconds - even for a small protein, this would require some 30 years of processing time for a single simulation. As such, the computing power and resources required to simulate protein dynamics places has seemed out of reach of all but the most powerful supercomputers.

The Folding@home project attempts to solve this computational problem using sheer numbers. Relying on the donated CPU time of hundreds of thousands of volunteers, it is currently operating at over 6.1 PetaFLOPS [4], and was the first computing project of any kind to surpass 1, 2, 3, 4 and 5 petaFLOPS. In comparison, the fastest standalone supercomputer in the world (the Cray Jaguar at the Oak Ridge National Laboratory) peaks at 1.75 petaFLOPs [5]. Using this volunteer distributed computing platform, the Folding@home project has recently accomplished an *ab initio* protein folding simulation for protein NTL9, which folds on the order of milliseconds.

2. Molecular Dynamics

The Folding@home project is fundamentally a molecular dynamics computation platform. In molecular dynamics, the motion of particles in a system is modeled using Newtonian mechanics. Given an initial position and a random velocity consistent with the desired simulation temperature, the interactions of all the atoms in a protein are approximated to determine the trajectory of each atom. This allows scientists to probe the motion of individual atoms in a way that would be impossible experimentally.

The interaction of molecules is modeled using a *force field* which describes the potential energy of a system of particles (i.e. atoms). A force field is comprised of two parts - a functional form calculating a system's potential energy as a sum of individual terms, and parameter sets for the unique values for every possible type of atom in a system. The functional form is comprised of two components: bonded terms consisting of bond, angle, and dihedral energies, and nonbonded terms consisting of long-range electrostatic and van der Waals forces.

Folding@home uses several optimized versions of GROMACS (GROningen MAchine for Chemical Simulations) to run its molecular dynamics simulations, which supports different force field models. One of the force fields that Folding@home primarily uses is AMBER (Assisted Model Building with Energy Refinement). The functional form of AMBER IS:

$$V(r^N) = \sum_{bonds} \frac{1}{2} k_b (l - l_0)^2 + \sum_{angles} \frac{1}{2} k_a (\theta - \theta_0)^2 + \sum_{torsions} \frac{1}{2} V_n [1 + \cos(nw - \gamma)]$$

$$+ \sum_{j=1}^{N-1} \sum_{i=j+1}^{N} \left\{ \varepsilon_{i,j} \left[\left(\frac{\sigma_{ij}}{r_{ij}} \right)^{12} - 2 \left(\frac{\sigma_{ij}}{r_{ij}} \right)^{6} \right] + \frac{q_i q_j}{4\pi\varepsilon_0 r_{ij}} \right\}$$

Bonds: The energy between covalently bonded atoms is approximated by a harmonic (ideal spring) force, which works best near the equilibrium bond length.
Angles: This represents the energy in non-equilibrium bond angles, again approximated by a harmonic force.
Torsions: This represents the energy due to twisting a non-single bond.
Double sum: The first part is a 12-6 Lennard-Jones potential, composed of Pauli repulsion (due to overlapping electron orbitals at short range, first term) and van der Waals force (attraction at long ranges, second term). The second part models electrostatic energy as a Coulomb potential.

There are a number of parameter sets for AMBER, with names beginning with “ff” and followed by a two-digit year number, such as “ff96”. Each parameter set contains values for atomic mass, van der Waals radius, and partial charge for individual atoms, and equilibrium values of bond lengths and angles. A particular atom may have different parameter values depending on the functional group it is associated with (for example, the oxygen in a carbonyl compared to a hydroxyl).

By minimizing the potential energy of a system, stable structures can be found. Because protein folding happens spontaneously, the lowest-energy conformation of a protein is usually its folded state. This allows us to simulate protein folding using a force field model of actual physics.

One force not yet addressed is the solvent that the molecule is in. Protein folding in living organisms occurs in water, and the interaction between the protein and water plays a key role in determining its final structure. For instance, in a folded protein hydrophobic amino acids tend to be in the center, shielded from the water solvent. There are two primary ways to model the effect of solvents on the system being modeled. In an explicit solvent model, solvent molecules are modeled by their individual atoms and bonds, the same as the protein itself. In an implicit solvent model, the solvent is represented as a continuous medium. Explicit solvent models intuitively seem a more accurate model since it makes fewer generalizations about the physics involved. However, there is a significant computational cost in including the individual solvent molecules in the simulation, and in a number of comparisons there was no increase in accuracy compared to the implicit model [6]. Folding@home uses a Generalized Born implicit solvent model for most of its simulations.

A typical molecular dynamics simulation will consist of thousands of such simulations at different temperatures, each with random initial velocities for the atoms consistent with the desired temperature. Although most of the trajectories will not result in a folded protein, some will. These trajectories constitute a simulated statistical sample of the folding rate of the protein. Assuming there is only one barrier between the unfolded and folded states of the protein, the fraction of trajectories that have folded can be modeled by the exponential distribution $\frac{n}{N} = 1 - e^{-kt}$, where n is the number of trajectories that have crossed the barrier, N is the total number of simulations, and k is the folding rate. For very short times where $t << 1/k$, this can be approximated by $\frac{n}{N} \approx kt$, and the rate can be estimated from the slop of a plot of n/N versus the time t. If this rate agrees with experimentally-derived values, it suggests that any conclusions drawn from this simulation represent the true microscopic details of folding in the system. However, research suggests that many proteins transition through more than two states while folding [7][8]. In such cases, the molecular dynamics trajectories can be clustered into states by similar kinetic behavior. Once clustered, the single exponential rates can be calculated between pairs of clusters, allowing for a comparative analysis of the rate-limiting effect of various transition states and pathways [9].

3. Volunteer Distributed Computing

Distributed computing is the use of multiple autonomous computer systems communicating through a network to solve a computational problem. Generally, a large, computationally unreasonable problem is divided into many smaller parts, each solved by an individual computer. An early example of such a system was the Distributed

Computing System (DCS) which was developed at the University of California, Irvine beginning in 1970 [10]. The potential of distributed computing grew exponentially with public access to the Internet. Starting in 1996 a number of volunteer distributed computing projects sprang into existence, including the Great Internet Mersenne Prime Search in 1996, distributed.net (attempting to crack RSA security and calculate optimal Golomb rulers) in 1997, SETI@home in 1999, and Folding@home in 2000.

The benefit Folding@home derives from distributed computing is not faster computations - in fact, the speed of an individual calculation is determined by the computer system whose time is being donated and likely to be run on an "average" consumer processor. Instead, it allows for thousands of trajectory simulations of the same protein to run in parallel, creating a large number of folding trajectories in much less time than a single computer would take.

The volunteer nature of Folding@home is a source of great complexity for the project, but also a great strength. Hundreds of thousands of non-scientists can contribute directly to current research with just their computer and an Internet connection. The perceived cost in donating is minimal, since both of these commodities are existing resources from the perspective of a volunteer. Researchers are not burdened with the cost of hardware, electricity, and maintenance associated with large centralized computing centers. There is also the advantage of a self-upgrading system - as volunteers upgrade their computers, Folding@home is likewise upgraded, allowing it to follow Moore's law without additional cost.

Because of the wide variety of hardware, driver, and software combinations volunteers may be using (Apple computers notwithstanding), significant effort is required to maintain compatibility with the Folding@home client software. Also, because even a modest improvement in performance over 400,000 hosts results in a huge jump in computational power, the assembly code for each processor type is hand-optimized to make the most use of the hardware. A striking example of this is the speed of the Folding@home client optimized for graphics processing units.

4. Graphics Processing Units

Graphics processing units (GPUs) are specialized processors meant for accelerating or offloading 2D or 3D rendering. Unlike CPUs, which typically have a few very fast processor cores, modern GPUs have a large number of slower processing units, allowing for tremendously powerful, massively parallel computation. For example, while high-end processors currently have at most four cores, a high-end GPU like the ATI Radeon 5970 can have 3200 processing units [11]. As such, a GPU-optimized version of the Folding@home client allowed for over 700 times improvement to the speed of its CPU-based implementations [12].

The GPU optimizations primarily dealt with memory bandwidth differences between GPUs and CPUs. For example, GPUs have very little cache to hide latencies from random memory access. Instead, related data must be stored in contiguous blocks. Repeating calculations can also be more efficient than storing/retrieving a value from memory. Similarly, communicating between the CPU and GPU across the PCIe bus can add latency greater than the time required to perform the calculation itself [12] . Minimizing function calls helped to alleviate this problem.

Because GPUs have typically been designed for a very specific function, they are considerably less flexible in the types of calculations they can perform compared to a CPU. For example, ATI boards lacked integer arithmetic and both ATI and NVIDIA only supported single-precision floating point numbers. Additionally, processors in GPUs are not independent of each other. Threads are arranged in groups of 16 to 64 on current GPUs, and all threads in a group must execute exactly the same instruction at the same time. Finally, ATI and NVIDIA use entirely different development tools. These additional complications make implementing the GPU clients far from a simple port, but given the dramatic potential speed increase the effort seems well-worthwhile. In fact, the NVIDIA GPU client currently contributes over half of the processing power of Folding@home, despite being less than 4% of the active clients [4].

Table 1. Benchmark Results (from [11])

Molecule	Atoms	Force field	Platform	Ns/day	Improvement
fip35	544	parm03	CPU/AMBER	4.5	-
fip35	544	parm03	ATI	279.2	62
fip35	544	parm03	Nvidia	576.2	128
Villin	582	parm03	CPU/AMBER	3.9	-
Villin	582	parm03	ATI	260.8	67
Villin	582	parm03	Nvidia	528.5	136
Lambda	1254	parm03	CPU/AMBER	0.79	-
Lambda	1254	parm03	ATI	141.7	179
Lambda	1254	parm03	Nvidia	201.6	255
α-spectrin	5078	parm99	CPU/AMBER	0.023	-
α -spectrin	5078	parm99	ATI	14.2	617
α -spectrin	5078	parm99	Nvidia	16.9	735

Improvement is the speedup obtained by running on the GPU versus running AMBER on the CPU.
ATI: Radeon HD 4870 GPU.
Nvidia: GeForce GTX 280 GPU.

As seen above, not only are the GPU clients running significantly faster than the CPU client, but they scale much better with the size of the protein being folded. While on the CPU the scaling is close to $O(N^2)$, where N is the number of atoms, the GPU clients scaled subquadratically in all cases and even sublinearly for ATI from 582 to 1254 atoms. Also, despite the lack of double precision floating point numbers, the accuracy of both the NVIDIA and ATI implementations compared favorably to other molecular dynamics algorithms, including those using double precision [11].

5. Simulation of Millisecond Folder NTL9 [13]

Due in large part to the performance gains provided by the GPU client, the Folding@home group successfully ran an *ab initio* protein folding simulation of the protein NTL9, which experimentally has a folding time of ~1.5 ms. This is longer than the previous slowest-folding (nanoseconds to microseconds) proteins by all-atom molecular dynamics simulations by orders of magnitude.

Trajectories were simulated at 300, 330, 370, and 450 K from native, extended, and random-coil configurations using the GROMACS GPU client. The AMBER ff96 force field with the GBSA solvation model was used. No productive folding trajectories were observed at 300 or 330 K, consistent with Arrhenius kinetics. 450 K trajectories exceeded the melting temperature of NTL9. The number of folding events n observed was consistent with a Poisson distribution $\langle n \rangle = \int M(t)ke^{-M(t)kt}dt$ reaching time t at the experimental folding rate k (640/s). The theory predicts on average ~1.8 folding trajectories, in agreement with the two found by simulation.

Markov State Models (MSMs) were used to describe the mechanical and kinetic aspects of folding. 100,000 microstate clusters were generated by an approximate k-centers algorithm (where each snapshot in the simulation is assigned to the closest of k clusters). These were then lumped into a 2000-macrostate model. The metastable states were diverse, with multiple possible folding pathways. The 10 pathways with the highest folding flux were calculated, which transited only 14 of the 2000 macrostates.

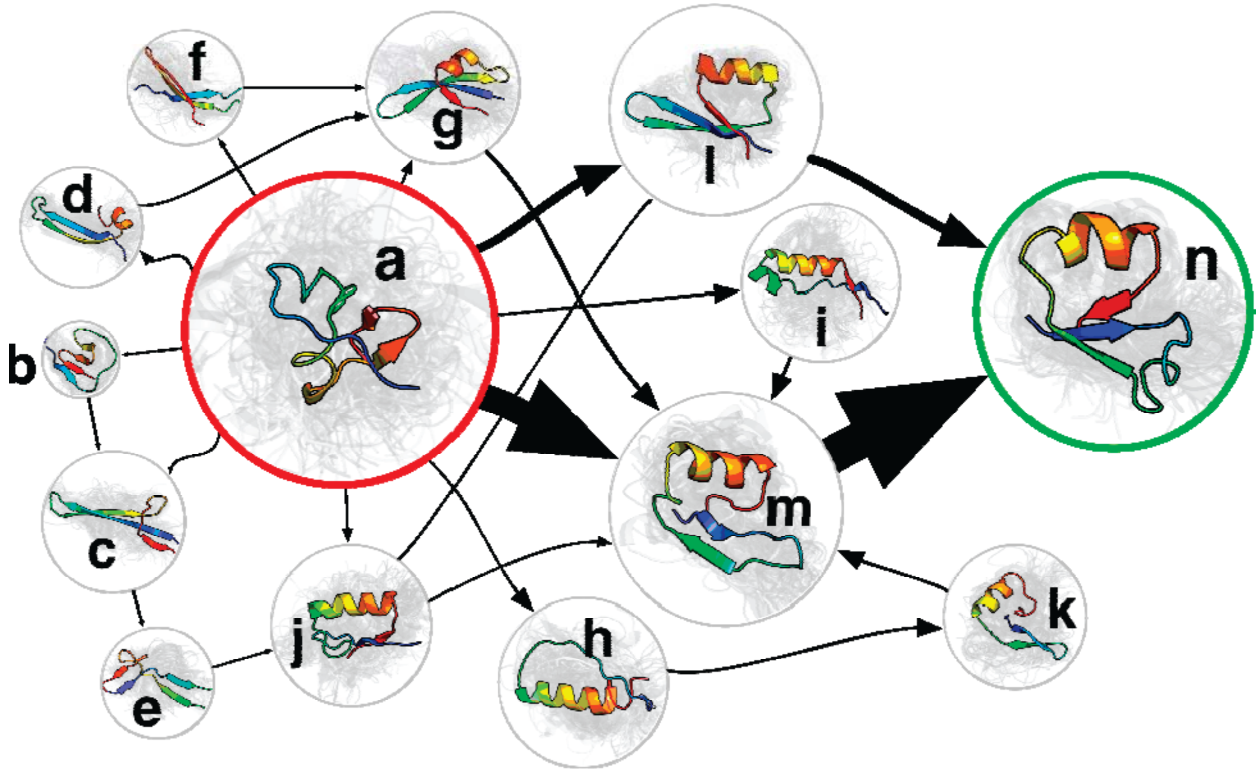

Figure 1. The 2000-state Markov State Model (MSM). These pathways account for only ~25% of the total flux and transit only 14 of the 2000 macrostates (shown labeled a-n). The visual size of each state is proportional to its free energy, and arrow size is proportional to the interstate flux. Modified from [13].

In order to determine the rate-limiting step in the folding reaction, three of the main structural elements of NTL9 were considered: the central helix (α), the pairing of strands 1 and 2 (β_{12}), and the pairing of strands 1 and 3 (β_{13}). The Q-value, a measure of native-like contacts, was calculated for each of these elements in relation to the p_{fold} value, a measure of the progress of a kinetic reaction. In general, Q values increase as p_{fold} increases, but the relative balance of Q_α, $Q_{\beta_{12}}$, and $Q_{\beta_{13}}$ varies. Specifically, only for macrostates with $p_{\text{fold}} > 0.5$ does notable β_{12} strand pairing occur. This suggests the formation of β_{12} is rate-limiting.

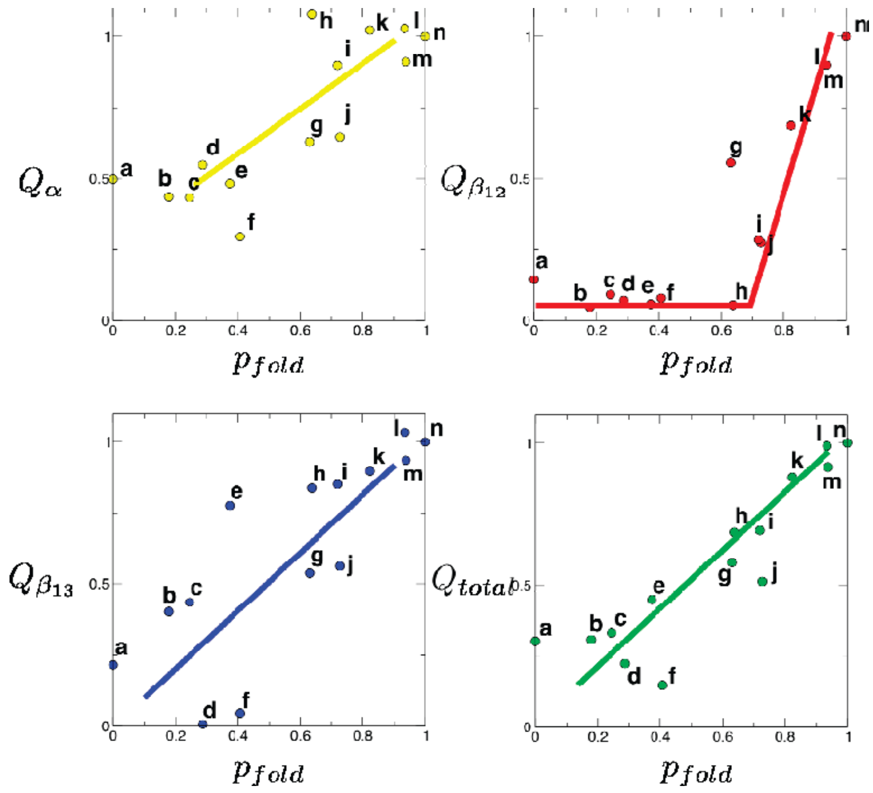

Figure 2. Q-values, which capture the extent of native-like structures, plotted versus pfold (committor) values. The lines are to guide the eye. Modified from [13].

6. Conclusion

The latest results from the Folding@home project represent an exciting advancement in our ability to fold proteins *ab initio* at much longer time scales than we could before. However, the entire model relies on a number of assumptions which may not be valid. In particular, the only validation between the simulation and experimental results is the reaction rate. This does not confirm or even suggest that the actual simulation trajectories reflect reality, only that the simulation folds at a statistically similar rate as experiments. That said, there is no reason to reject the findings on this basis. Without a direct way to observe protein folding, molecular dynamics simulations remain one of the best ways to elucidate the mechanisms behind such events. Regardless of the validity of this one particular protein folding, the power harnessed by distributed computing using GPU clients is undeniable. Folding@home is the single most powerful computational engine currently in existence, a fact that is difficult to argue with.

References

[1] D. Selkoe (2003). "Folding proteins in fatal ways". Nature, 426: 900–904.
[2] F. Chiti, C. Dobson (2006). "Protein misfolding, functional amyloid, and human disease", Annual Review of Biochemistry, 75: 333–366.
[3] M. S. Friedrichs, et al (2009). "Accelerating Molecular Dynamic Simulation on Graphics Processing Units", Journal of Computational Chemistry, 30(6): 864-872.
[4] "Client statistics by OS". Folding@Home. 06-04-2010. http://fah-web.stanford.edu/cgi-bin/main.py?qtype=osstats. Retrieved 06-04-2010.
[5] "TOP500 List - June 2010". TOP500 Supercomputing Sites. 06-2010. http://www.top500.org/list/2010/06/100. Retrieved 06-04-2010.
[6] C. D. Snow, et al (2005). "How well can simulation predict protein folding kinetics and thermodynamics?" Annual Review of Biophysics & Biomolecular Structure. 34:43-69.
[7] M. Karplus, D. L. Weaver (1979), "Diffusion-collision model for protein folding", Biopolymers, 18:1421-1437.
[8] W. Y. Yang, M. Gruebele (2004), "Detection-dependent kinetics as a probe of folding landscape microstructure", Journal of the American Chemical Society, 126(25):7758-7759.
[9] A. L. Beberg, D. L. Ensign, G. Jayachandran, S. Khaliq, V. S. Pande (2009). "Folding@home: Lessons from eight years of volunteer distributed computing", ipdps, pp.1-8, IEEE International Symposium on Parallel & Distributed Processing, 2009
[10] D. J. Farber, K. Larson (1970), "The Architecture of a Distributed Computing System - An Informal Description", University of California, Irvine, CA, Technical Report Number 11
[11] "ATI Radeon™ HD 5970 Graphics Feature Summary". AMD.com. http://www.amd.com/us/products/desktop/graphics/ati-radeon-hd-5000/hd-5970/Pages/ati-radeon-hd-5970-specifications.aspx. Retrieved 06-05-2010.
[12] M. S. Friedrichs, et. al. (2009). "Accelerating Molecular Dynamic Simulation on Graphics Processing Units", Journal of Computational Chemistry 30:864-872.
[13] V. A. Volez, et al (2010). “Molecular Simulation of ab Initio Protein Folding for a Millisecond Folder NTL9(1−39)”, Journal of the American Chemical Society 132(5): 1526–1528