Folding@home: Lessons From Eight Years of Volunteer Distributed Computing

Adam L. Beberg[1], Daniel L. Ensign[2], Guha Jayachandran[1], Siraj Khaliq[1], Vijay S. Pande[2]
[1]*Computer Science Dept, Stanford University*
[2]*Chemistry Department, Stanford University*
{beberg@cs., densign@, guha@cs., siraj@cs., pande@}stanford.edu

Abstract

Accurate simulation of biophysical processes requires vast computing resources. Folding@home is a distributed computing system first released in 2000 to provide such resources needed to simulate protein folding and other biomolecular phenomena. Now operating in the range of 5 PetaFLOPS sustained, it provides more computing power than can typically be gathered and operated locally due to cost, physical space, and electrical/cooling load. This paper describes the architecture and operation of Folding@home, along with some lessons learned over the lifetime of the project.

1. Introduction

Protein folding is a fundamental process of biology in which proteins self-assemble or "fold." Many diseases (such as Alzheimer's, Parkinson's, and BSE) are related to situations where protein misfolding and aggregation occurs, disrupting ordinary cellular functions. However, simulations of protein folding and dynamics are computationally challenging due to the combination of the great detail needed in the simulations coupled with the long timescales required to compare with experiment.

The computational power needed to simulate protein dynamics on the timescales needed to observe folding is far beyond what modern computers can provide in reasonable time. A fast single-core PC may be able to do on the order of 20 nanoseconds of simulation per day for a small protein, and a Cell processor[1] can simulate as much as 500 nanoseconds per day for small proteins, but most important protein dynamics occur on timescales from milliseconds to seconds. To bridge this timescale gap, we must turn to distributed computing, multi-core, and SIMD (single instruction, multiple data) methods.

Volunteer computing is a popular term for distributed computing where the work is done on computers recruited from people on the Internet, not on machines under the control and management of the project. This approach can provide the necessary computational power for bridging the time scale gap, but it introduces many new problems with respect to security, reliability, and coordination.

The architecture of Stanford University's Folding@home project [2] [3] additionally involves a combination of load balancing, result feedback, and redundancy that are only required on volunteer systems of this scale.

2. Related work

The Distributed Computing System (DCS) [4] [5] (1970-77) represents the first work where computers had the capabilities to do distributed computing on a network scale, and tackled all the main issues we continue to face with distributed computing today. While the hardware and networking have improved with Moore's law, and the number of users on the Internet has grown by many orders of magnitude, the goals of distributed computing have not changed significantly in 30 years. What has changed greatly is the loss of any assumption of reliability and the decay of the security environment, which introduce an additional set of challenges.

The ability to use distributed computing outside of academic networks, and thus build projects larger than several hundred machines by recruiting volunteers, did not emerge until the 1990s. The first large-scale projects to do this and gain public appeal were prime number hunting by the Great Internet Mersenne Prime Search (GIMPS) [6], and encryption cracking efforts by distributed.net [7] and DESCHALL with the motive of getting the U.S. encryption export restrictions removed. All three appeared around the same time in 1996-97 as the public began to access the Internet.

978-1-4244-3750-4/09/$25.00 ©2009 IEEE

A second wave of projects began in 1999 with SETI@home [8] which searches for signs of intelligent extraterrestrial life in radio signals, and Folding@home (2000), the topic of this paper. During this wave volunteer computing took off, and now there are hundreds of active projects with a wide range of sizes and goals.

3. Scientific goal of Folding@home

The major use of the Folding@home infrastructure is statistical calculation of molecular dynamics trajectories for models of biological systems. In molecular dynamics, Newton's equations of motion are integrated for each atom in the system in order to propagate its trajectory, given an initial position and a random velocity chosen to be consistent with the desired temperature. The distributed parallelism of Folding@home is not used to speed individual calculations but rather to obtain hundreds or thousands of parallel simulations. These trajectories then constitute a statistical sample of simulations, some of which may fold but (depending on the details of the simulations) most do not.

More trajectories will fold as the length of individual trajectories increases, but a few folding events in thousands of short trajectories are sufficient to estimate the rate of folding. If there is only one barrier between the unfolded and folded states, then the fraction of trajectories that have crossed the barrier is given by a single exponential $n/N = 1 - e^{(-kt)} \approx kt$, where n is the number that cross the barrier and N is the total number of simulations, and k is the folding rate. The approximation holds for very short times $t << 1/k$. The rate at which barrier crossing occurs can therefore be estimated from the slope of a plot of n/N versus the simulation time t. Calculation of the relevant rates provides the best (and often only) comparison possible to dynamical experiments. If the rate is consistent with experiment then it is more plausible that the microscopic details such as the order of events in any folding trajectory represent the process of folding in the system.

However, several recent studies such as [9] and popular models of protein folding [10] indicate that many systems should fold through more than two states, so the above analysis is incomplete. When the kinetics are more complicated than single exponential, different states can be identified in the Folding@home simulations and single exponential rates can be calculated between each pair of states. Procedures for state identification and rate calculation are productive and ongoing areas of research in many groups [11].

In order to build such models, the conformations sampled from molecular dynamics trajectories must be clustered into states. This is often, but not always, accomplished, by k-medoid type algorithms which assigns each snapshot in the simulations to the closest of k generators; usually new generators are chosen from these initial clusters and the procedure is iterated to convergence. Once such clusters or "microstates" are identified, they can then be lumped together into macrostates, which are groups of states characterized by similar kinetic behavior. The conformations in a given macrostate convert to different macrostates at the same rates. The resulting model consists of a matrix of rates or transition probabilities; it incorporates all the kinetics of the system, as the eigenvalues are related to the rates of population flux in the system, and it can be propagated out to infinite time in order to calculate equilibrium thermodynamic properties.

Because the eigenvalues of such matrices correspond to important kinetic properties, it is often important to improve their precision. Each element of the rate matrix can be efficiently tested for its impact on the uncertainty of an eigenvalue of interest [12]. This calculation therefore predicts what additional simulations would be required to decrease the uncertainty of that interesting eigenvalue.

Because these calculations are so efficient, they can be applied as part of an adaptive sampling algorithm for Folding@home simulations. First, a set of simulations are run to obtain initial estimates of the states and the interconversion probabilities (or interconversion rates). Then, the matrix can be tested with regard to the uncertainty of one eigenvalue (perhaps the most uncertain eigenvalue, or an eigenvalue of special interest because of its association with interesting kinetic processes) to propose a set of new simulations to run on Folding@home. Once this new data is collected, a new rate matrix can be quickly rebuilt. The testing procedure can then be repeated until the uncertainty is reduced to acceptable levels. At any point, the clustering may also be recalculated, although this takes much more time than the uncertainty tests so must be done more sparingly.

The correct elucidation of the states and a well-estimated matrix of transition probabilities, generated by the adaptive sampling procedure just described, produces a model which represents all the kinetics and thermodynamics of the protein folding process for the system under consideration. This model (derived from the behavior of molecular dynamics trajectories in an empirical force field) can therefore generate specific, testable hypotheses regarding the order of folding events, the impact of point mutations, and the validity of the empirical force fields themselves.

4. Architecture

Folding@home uses a client-server architecture depicted in Figure 1, with arrows showing the payload flows. First, a client that has been installed by a volunteer asks the assignment server to assign it to a work server (1). Next, the client talks to a work server and requests a work unit, which is a set of input files needed to work on a job for a length of time (2). Based on the work unit, the client may then download the computational core required to do the work from a web server (3). Once the work is completed, the client sends the results to the same work server, or to a specially-designated collection server if the work server is unreachable (4). Log files and credits are then collected from the work servers and passed to the statistics server for tabulation and display to the participants (5). While Folding@home has concentrated on protein folding simulations, this architecture is extremely flexible and could support many other types of projects.

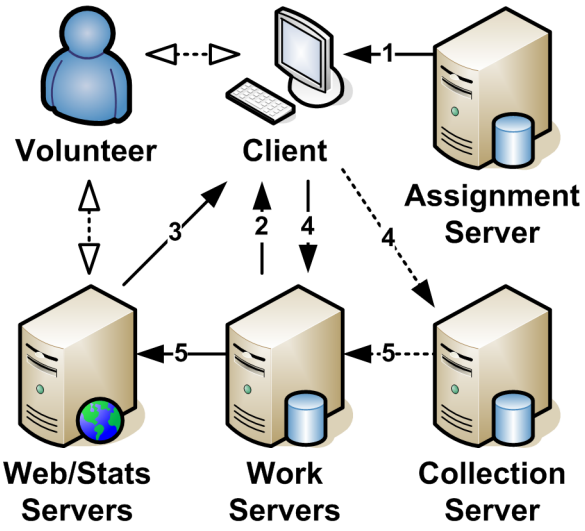

Figure 1. Folding@home architecture. Arrows depict data flow.

4.1. Clients and computational cores

The Folding@home clients serve two roles. The main role is a loop to get one or more work units, process them, and return the results without interfering in the operation of the machine. The other role is to interface with the volunteer keeping them informed of what is happening on their computer.

Each work unit contains a header that tells the client which core type and version is needed to process it, and where this core can be downloaded from. A core is downloaded as a cryptographically signed binary for that platform. By separating the client from the cores, we gain the ability to do any kind of computation as well as upgrade the cores securely without having to reinstall any software.

The most frequently used cores in our group are based on the molecular dynamics simulation package GROMACS [13] which contains a great deal of hand-optimized assembly code for each CPU type allowing it to take full advantage of the available hardware. Even a small speedup over 400,000 hosts represents a large amount of computational power, which would justify the effort of hand optimization, and often the improvements are quite dramatic. The PowerPC version uses AltiVec while the x86 version has SSE2 support for many of the innermost loops. This is fairly common for scientific code, so unlike the client and server (which are portable), almost any scientific core will have portability limitations without rewrites. This is most dramatically illustrated during the porting of the scientific core to the GPU and the IBM Cell processors, which required complete algorithm redesigns to run on the different parallel architectures.

In addition to their primary roles in scientific computation, cores must also perform checkpointing by saving the exact state of the calculation to disk periodically, because a volunteer's PC may crash or be turned off at any time. Since work units can run for days, checkpointing reduces the amount of work that is lost to several minutes when the client resumes operation. Most applications do not normally include checkpointing, or do so without the complete state needed, so adding this capability is the most significant part of making code that is designed to run on a single system run in a distributed system.

The effort involved in porting, maintaining, and support of the cores forces a focus on a small number of platforms. We must choose platforms that have a large number of deployed units and that can perform well on the desired types of computation. The desktop and server marketplace has consolidated to Windows, OS X, and Linux with all three now focused on the x86 architecture, allowing us to use the same assembly code in cores. We also support OpenBSD and FreeBSD through automatic branding of the cores by the Linux client, which makes the software think it is running on a Linux machine. The GPU, Cell, and other multi-core architectures like Intel's announced Larrabee architecture vary greatly in programming model and still lack the mature development tools and debuggers needed to easily develop for them. Thus, development on these exotic architectures requires a large investment of software engineering time.

The client's role in volunteer interface and promotion is as critical as portability. These concerns

require additional effort to provide features that volunteers want to see, bundle up installers, and make the client easy to use by non-technical volunteers. Installed software may not be updated regularly or at all and may remain running for years without user maintenance. This makes the separation of the client into a main client and an updateable core very important to be able to continue to advance the science which evolves rapidly.

Volunteers can set a number of preferences that one needs to respect. Some choose to only do small work units because their Internet connection is charged by the amount of bandwidth used. Others connect by modem and need the client to wait for the user to connect to the Internet. Issues such as the priority the core runs, what percentage of the CPU we should use to avoid overheating, and stopping when a laptop is unplugged are just a few other examples.

Folding@home was started shortly after the time that firewalls and NAT began to be commonly used and made clients connecting to Stanford directly difficult. All traffic travels over the same ports as the Web uses, ports 80 and 8080. While this allowed us to pass through firewalls, it also created the need to deal with web proxies that were not coded to the official web standards and many different Windows TCP stacks. While the situation is now much more stable, we still run into new problems as our users install new network setups.

The OpenGL graphics, originally integrated into the client but now a separate source-available application, shows the volunteer the protein being simulated. While this may seem like a waste of resources, it is important to many volunteers to see what is happening on their computer, and helps in recruiting.

4.2. Work Servers

The primary role of a work server is to generate work units for its specific project(s), and to accept, store, log, and analyze completed work. The details of each of these steps, and the level of feedback from results to new work, can vary greatly by project. Generating work units and analyzing what work to do next to best improve the model can be an intensive process for biomolecular simulations, where previous results are used to generate subsequent work. This planning represents a majority of the computation done by a server.

Each work server can run many projects and is administered by a different researcher. This allows servers to be maintained when needed and reduces conflicts over drive space and other resources. This is especially important because, as of February 2009, there were 70 separate work servers in Folding@home running hundreds of active projects. The total storage needed for the data resulting from Folding@home has passed 350 terabytes plus additional offline archives, with a similar amount of off site backups.

4.3. Assignment servers

The assignment servers act as the global scheduler and load-balancer in the Folding@home system. An assignment server is contacted first by the client whenever work is needed and its address is the only one that needs to be hard-coded into the clients.

There are many factors that determine project priority, including a determination of the most time-critical results, which results need the most CPUs, as well as publication deadlines. Moreover, many client-side parameters are considered in making assignments, including client reliability, preferences for work type (e.g., beta testing, packet size), measured upload and download bandwidths, memory capabilities, and client version [14].

The assignment server monitors all of the work servers to see which are online along with what CPU and operating system they can provide work for. For example, if an x86 machine running OS X requests an assignment then it will be assigned to a work server that has work units available that will run on x86/OS X. From among available servers, and taking into account the priorities, server load, and amount of remaining work, the assignment server directs the client to a server along with a token to allow it to get a work unit.

4.4. Collection servers

The work servers are reasonably stable individually but due to the fact that they are scattered around the Stanford campus and at other collaborating locations, server downtime from power and network outages are to be expected. Hardware failures, while individually rare, occur frequently with any large number of servers.

Collection servers were introduced into the system in version 5.0 of the clients in 2004, and have been the only major architectural addition. When a client cannot upload the results of a work unit to the originating work server, it instead returns the work to the collection server. If the work unit is on the list of send work units collected from the servers, it is then accepted, credited, and eventually passed along to the work server once it comes online.

This allows the client to continue to make progress even when the servers are down, as they can send the results to the collection server and request more work

from a live work server. When the server returns, it is not flooded with requests from waiting clients, which could result in a denial of service.

4.5. Statistics, Forum, and Website

The user statistics website is in many ways the heart and soul of any distributed computing project, because it allows volunteers to see how much they are contributing. The statistics server is a large database storing every result ever received by Folding@home and the credits (points) assigned to each user and team. Log files from all of the work servers are gathered up periodically and the completed work is credited to the volunteer and team for public display.

The forums, which are run by dedicated volunteers, offer users of Folding@home both technical support and a sense of community. We also get a great deal of helpful feedback on the forums that allows us to learn what features users want and what problems they are having.

The website is the central location for all of the information on Folding@home, and includes news, answers to frequently asked questions, results, and papers. The website also hosts an educational section put together by Tug Sezen to facilitate teaching high school students and the public about the science and mechanisms behind protein folding.

5. Volunteer computing

Simultaneously the most significant advantage and greatest complexity of any volunteer computing initiative, such as Folding@home, is the reliance on volunteers to setup and run clients. Having hundreds of thousands of computers on the Internet allows us to scale Folding@home to 100-1000x what could be done locally. This allows us to tackle scientific calculations that we could not otherwise do for reasons of cost, space, overhead and operational complexity.

The main challenge of large local computation is the cost for the hardware, electricity, cooling, floor space, and system administrators. By using distributed computing, we can eliminate many of these costs completely because we use existing resources. The cost of the volunteer's computer and Internet connection is already spent since they need a computer for other reasons. This makes the only additional cost the difference in electrical use of the busy system versus idle ones. This per-volunteer cost has been considered insignificant by those participating, but may become a concern if electrical costs increase greatly in a geographic area.

The management of clients by volunteers gives us the advantage of a self-upgrading system. As volunteers upgrade their systems, Folding@home is upgraded in capacity as well. As a whole this gives us Moore's law growth without any additional effort. As new architectures are released such as the recent move form PPC to x86 by Apple or the move from 32 to 64bit Linux, we have to do the work of porting and testing the cores and recompiling the client, but the computers are upgraded over time without our intervention.

Windows and Linux present a near-infinite combination of hardware, software, and drivers that would not be encountered in a local setting. This means that a significant amount of time is spent dealing with incompatibilities when the clients are developed, and every time a new version of the operating system is shipped such as Windows 7, or the latest version of a Linux distribution.

In exchange for the use of their computers, volunteers want to see something in return. In the case of Folding@home this is centered around the protein folding research yielding results in peer-reviewed publications, with the goal of gaining insight into how a class of diseases work by studying the fundamental physics and chemistry processes involved. Efforts that do not yield results that can benefit the volunteer in some meaningful way now or in the future cannot appeal to a wide audience. All of the large volunteer computing projects are currently either run by or directly associated with well known research universities.

The statistics system also allows volunteers to compete against each other individually and in teams with the credits assigned for completing work units, feeding the primal need to compete against others. This aspect drives recruitment of new volunteers by spreading word of mouth and other forms of viral marketing. Volunteers are often fiercely loyal to projects like Folding@home once they get involved, and many have team websites, recruiting efforts, and even merchandise of their own. In an effort to boost their statistics they often upgrade their hardware more aggressively than those not involved in volunteer computing. This serves as an additional positive feedback mechanism for the self-upgrading aspect of the system.

The statistics system also comes with a unique set of challenges in that some very small fraction of the volunteers will attempt to cheat to boost their statistics. The most common form of cheating we have encountered is volunteers installing the clients on machines they do not own at school or work in an effort to increase their personal or team statistics. Worse are the use of Trojan horses on P2P file sharing systems to install the client and gain in the statistics. Both are strongly discouraged and easily noticed by

system administrators or by people noticing rapidly rising users and teams, so that users' statistics are zeroed and they are blacklisted.

Being volunteers, participants can leave the project at any time. New clients are installed at a slightly higher rate then they are lost, depending on the level of press coverage and result publication rates. The activities of other projects seem to have little effect on participation during new releases or press coverage events. The number of people participating in volunteer computing compared to the number of users on the Internet is insignificantly small, so new projects can still grow without cannibalizing others.

Managing volunteer feedback in forums and other forms can be a large added burden on the researchers who run the projects if it is not handled correctly. A great deal of time needs to be spent helping people get clients running and responding to questions. Most often those in need of help are new volunteers, who are unfamiliar with how to run the clients or find answers to their questions. In this case, we have received the benefits of another aspect of volunteer computing: the most committed volunteers serve as forum moderators, write 3rd party software add-ons, debug problems, answer new user questions, and do a great deal to promote Folding@home. Any new information tends to propagate through the team sites and forums very quickly, which makes information even more widely accessible. Without these super-volunteers, projects would be impossible to run because all time would be devoted to development or support activities.

In the last few years, due to public awareness of concerns over the energy use of computers left on and their effects on electric bills and global warming, there is a very significant pressure towards projects that have direct scientific impact and measurable positive influence on the human condition. This is another social dimension in which projects must compete, and can only serve to improve the quality of projects in the future. A critical issue here is that while volunteers are a powerful asset to have, they require time and energy to maintain and keep happy with the project.

Frameworks for developing further distributed computing projects have been developed with a large variety of design goals. distributed.net and Folding@home both use the Cosm [15] tools. Cosm has a set of libraries and toolkits to develop distributed computing projects, and dates back to 1995. The Berkeley Open Infrastructure for Network Computing (BOINC) [16] now used by SETI@home and many others, launched in 2002. BOINC provides a standard client, sever, and statistics system, but with this fixed architecture comes limitations on the types of projects it can accommodate. Condor [17] is an idle-time harvesting system that is designed for campus-scale networks and has been in use since 1988. It is designed to require little or no changes to the user's software making it simple and an option for those without access to their application's source code.

6. Folding@home today

We currently support the seven platforms listed in Table 1. The table lists the number of machines that have been active by returning work recently, and the number of teraflops reported by the cores. We use very conservative estimates for our teraflops ratings based on the actual speed of the code, not the theoretic peak for the chips. We also only count actively participating clients.

Table 1. Approximate Folding@home Host Statistics (February 2009)

Client Type	TFLOPS	Clients
Windows x86/x64	275	280,000
Mac OS X ppc	5	6,000
Mac OS X x86	25	9,000
Linux x86	50	28,000
ATI GPU	1,125	10,000
NVIDIA GPU	1,900	17,000
PLAYSTATION 3	1,400	50,000
Total (approximate)	**4,800**	**400,000**

Multi-core PCs are now pervasive, with 4-core chips available in commodity PCs. This has led us to make all of our new computational cores aware and able to take advantage of this resource. Version 5.91 and up support multiple CPUs and cores on all platforms using an MPI-aware version of the core and a helper program. Preliminary results show superlinear scaling when these methods are used.

The GPU client has been available since late 2006, when supporting more than a handful of specific video cards was difficult because of variation in the architectures even from the same manufacturer. Support for high precision IEEE floating point was and is also not perfect. Originally, due to the lack of compilers, debuggers, and documentation, programming for the GPU was extremely difficult. An additional challenge was getting a set of computational cores and drivers that function well together for a set of cards that volunteers owned in large numbers. While these were significant obstacles, they have been mostly overcome as the tools, drivers, and hardware have matured.

The PlayStation 3 (PS3) client was launched in March 2007, and has grown to over 50,000. Because of the high floating point performance of the Cell processor, and a large amount of support and

promotion by Sony, these machines were for many months after release contributing the largest portion of Folding@home's raw speed.

The GPU clients are the fastest, but the most limited in terms of both ease of development and scope of accurately-computable simulations. The PS3 is limited by the architecture that allows only certain types of algorithms to perform well, and they are never upgraded but this makes development easier since there are no changing or unknown elements. Current CPU and multi-core CPUs are the most flexible in supporting any algorithm, some of which are critical, but have become many times slower in terms of raw speed.

Even with complete flexibility, the machines available change over time and may be nothing like what is predicted. The programming and porting never ends, and Moore's Law marches on.

7. Conclusions and future work

Folding@home has passed 2,000 projects that have been coordinated by a wide variety of researchers both at Stanford and collaborators elsewhere. Over 60 peer-reviewed publications have resulted from the processing power that has been harnessed, and Folding@home is currently the fastest computer system ever constructed in terms of effective FLOPS.

Folding@home has been an incredible resource for the Pande group and its collaborators to do computational chemistry on a scale impossible by any other means. However, there are still significant challenges in doing distributed computing. One of the most significant is the backend requirements. We maintain a large server cluster with a great deal of storage (hundreds of terabytes) to handle, store, and post-process Folding@home results. Since scaling to more clients is now limited by the backend rather than the algorithms employed, eliminating this bottleneck is a central goal for future work.

One effort in this direction is Storage@home [18] which will allow for distributed storage, and will be released and integrated along side Folding@home. This will allow us to leverage both storage and computation, combining the two to do rapid post-processing of the results generated by Folding@home. In the future, we will combine storage and analysis in a volunteer computing mechanism to help eliminate parts of the backend bottleneck and greatly increase our ability to ask questions.

Another ongoing problem is the constantly changing platforms on which we run, some of which have proved to be somewhat unreliable even when running at factory specifications. Thalweg [19] is being developed to address this and isolate the main code base from the details of the hardware in a more portable way, and provide measures to prevent soft errors (memory corruption and logic errors) from going unnoticed.

Finally, it is often asked why people contribute to volunteer computing. One benefit of a long running project is the accumulation of data in this regard as well, and the sociological aspects of volunteer computing in Folding@home will be studied in more detail in the future.

8. Acknowledgements

The original 1.0 version of Folding@home was written by Vijay Pande and Jarrod Chapman, based on the Cosm Client-Server SDK, with implementation advice from Adam Beberg (who continued to help throughout the years of the project); the protein visualizer was Chapman's work and remains in the client today and a rudimentary statistics system and web site was created by Pande. Aspects of the Windows GUI framework which remain today were written by Beberg.

Version 2.0 was architected by Siraj Khaliq and Vijay Pande to include the concept of a client-core separation (Khaliq) and an assignment server (Pande); a new statistics system was created by Chris Snow as well. Michael Shirts was instrumental in the Tinker core (version 2 and later) and GROMACS core (version 3 and later). Erik Lindahl has given instrumental advice for the GROMACS core since its incorporation into Folding@home.

Guha Jayachandran was instrumental in the more recent client releases (versions 3.1-6), and modifications to the servers. Recent years brought new cores, including V. Vishal's GPU core, and Young Min Rhee's AMBER and CPMD cores. Rhee also contributed server code updates, and the collection server code. Tug Sezen made several contributions to the web site for educational purposes. Developers, lead by Noam Rimon, at Sony Computer and Entertainment America wrote the PS3 client, in collaboration with Pande, Jayachandran, and Vishal. Peter Kasson implemented GROMACS 3.3.1 as a core for both single processor machines, as well as an MPI implementation for multi-core machines. The statistics system was updated in version 5 by Vishal and, more recently, Del Lucent has updated the statistics system for version 6.

We are indebted to the Folding Community Forum moderators who answer limitless volunteer questions and track down bugs, all the people that run teams, and the people that helped code the Cosm libraries we use. We also thank the Pittsburgh Supercomputing Center for the use of their backup facilities.

Finally, we thank NSF and NIH for their support of this work.

9. References

[1] E. Luttmann et al., "Accelerating Molecular Dynamic Simulation on the Cell processor and PlayStation 3", Journal of Computational Chemistry, vol. 30, (2), pp. 268-274, 2008.
[2] M. R. Shirts, V. S. Pande, "Screensavers of the world unite!", Science. 290:1903-1904 (2000).
[3] S. M. Larson, C. Snow, V. S. Pande, "Folding@Home and Genome@Home: Using distributed computing to tackle previously intractable problems in computational biology," Modern Methods in Computational Biology, R. Grant, ed, Horizon Press (2003).
[4] David J. Farber, K. Larson, "The Architecture of a Distributed Computer System - An Informal Description," University of California, Irvine, CA, Technical Report Number 11, September 1970.
[5] Paul V. Mockapetris, David J. Farber, "The Distributed Computer System (DCS): Its Final Structure," University of California, Irvine, CA, Technical Report, 1977.
[6] The Great Internet Mersenne Prime Search (GIMPS). 1996. [Online]. Available: http://www.mersenne.org/
[7] The distributed.net website (DCTI). 1997. [Online]. Available: http://www.distributed.net/
[8] The SETI@home website. 1999. [Online] Available: http://setiathome.berkeley.edu/
[9] W. Y. Yang, M. Gruebele, "Detection-dependent kinetics as a probe of folding landscape microstructure", Journal of the American Chemical Society, vol. 126, (25), pp. 7758-7759, June 2004.
[10] M. Karplus, D.L. Weaver, "Diffusion-collision model for protein folding," Biopolymers, vol. 18, pp. 1421-1437, 1979.
[11] F. Noe, S. Fischer, "Transition networks for modeling the kinetics of conformational change in macromolecules," Current Opinion In Structural Biology, vol. 18, (2), pp. 154-162, 2008.
[12] N. Singhal Hinrichs, V. S. Pande, "Calculation of the distribution of eigenvalues and eigenvectors in Markovian state models for molecular dynamics," Journal of Chemical Physics, vol. 126, (24), 2007.
[13] The GROMACS website. 1993. [Online] Available: http://www.gromacs.org/
[14] G. Jayachandran, "Parallel Computing Methods for Probing Biomolecular Kinetics and Thermodynamics" (Ch. 2). Stanford University. 2007.
[15] The Mithral website (Cosm). 1995. [Online]. Available: http://www.mithral.com/
[16] D. P. Anderson. "BOINC: A System for Public-Resource Computing and Storage," 5th IEEE/ACM International Workshop on Grid Computing, pp. 365-372, Nov. 8 2004, Pittsburgh, PA.
[17] M. Litzkow, M. Livny, M. Mutka, "Condor - A Hunter of Idle Workstations", Proceedings of the 8th International Conference of Distributed Computing Systems, pages 104-111, June, 1988.
[18] A. L. Beberg, V. S. Pande, "Storage@home: Petascale Distributed Storage", Proceedings of the 2007 IEEE International Parallel and Distributed Processing Symposium (IPDPS 2007), Long Beach, CA, March 2007.
[19] (2009) The Thalweg website. [Online] Available: http://www.OpenSIMD.com/