

Homogeneous Redundancy: a Technique to Ensure Integrity of Molecular Simulation Results Using Public Computing

M. Taufer^{1,2,3}, D. Anderson⁴, P. Cicotti⁵, C.L. Brooks III^{1,2}

¹ Dept. of Molecular Biology (TPC6) ² Center for Theoretical Biological Physics
The Scripps Research Institute University of California, San Diego
La Jolla, California 92037, U.S. La Jolla, California 92093, U.S.

³ Dept. of Computer Science ⁴ Spaces Sciences Laboratory ⁵ Dept. of Computer Science
University of Texas, El Paso University of California, Berkeley University of California, San Diego
El Paso, Texas 79968, U.S. Berkeley, California 94703, U.S. La Jolla, California 92093, U.S.

mtauffer@utep.edu, davea@ssl.berkeley.edu, pcicotti@cs.ucsd.edu, brooks@scripps.edu

Abstract

Distributed computing using PCs volunteered by the public can provide high computing capacity at low cost. However, computational results from volunteered PCs have a non-negligible error rate, so result validation is needed to ensure overall correctness. A generally applicable technique is "redundant computing", in which each computation is done on several separate computers, and results are accepted only if there is a consensus. Variations in numerical processing between computers (due to a variety of hardware and software factors) can lead to different results for the same task. In some cases, this can be addressed by doing a "fuzzy comparison" of results, so that two results are considered equivalent if they agree within given tolerances. However, this approach is not applicable to applications that are "divergent", that is, for which small numerical differences can produce large differences in the results.

In this paper we examine the problem of validating results of divergent applications. We present a novel approach called Homogeneous Redundancy (HR), in which the redundant instances of a computation are dispatched to numerically identical computers, allowing strict equality comparison of the results. HR

has been deployed in Predictor@home, a world-wide community effort to predict protein structure from sequence.

Keywords: *Replication Techniques, Heterogeneous Computing, Monte Carlo Simulations, Molecular Dynamics Simulations.*

1 Introduction

The interest in computer simulations of molecular phenomena has grown in the last decade. At the same time, the study of these phenomena has expanded from supercomputers to distributed computers, and ultimately to public computing [1, 2, 3, 4, 5]. In the latter approach, computations are done on PCs belonging to volunteer "participants" all over the world.

Computational errors or variations in computed results are a significant issue in public computing. These arise from several sources:

Hardware malfunctions: Many participants in public computing projects modify their PCs by increasing the clock rate. This "overclocking" can cause hardware bit errors. These errors may occur only in floating-point calculations, so that the computer generates incorrect results but does not crash. Moreover, such errors may occur sporadically, for example because of fluctuations

in ambient temperature.

Incorrect software modifications: A number of public computing projects make their application source code available, allowing participants to port it to new platforms, to modify and to recompile it so that it runs faster on particular architectures. These modifications may produce deviations in the output.

Malicious attacks: A small number of participants modify or replace the client software so that it returns incorrect results. These attackers are typically motivated by getting "credit", improving their position on the ranked list of participants.

Techniques have been proposed for dealing with errors. For example, the client can periodically run a function that checks numerical correctness. If an error is found, the user is notified and the current computation is abandoned. This is done in distributed.net [6] and Folding@home [3, 4]. Other techniques have been developed [7] that allow the project to verify, by examining a returned result, that the client has performed the entire calculation. This removes the "credit" incentive for attackers. However, neither of these techniques solves the problem in general. It is also worth noting that sandboxing techniques, such as used in XtremWeb [8], while preventing damage from misbehaving applications, do not solve the problem of computational errors.

A general technique for dealing with errors or result variations is "replicated computing". In this approach, the same computation is performed on different PCs, and the results are compared. If, for example, each computation is done three times, and two out of three results agree, we might accept these two results as being "correct". If we ensure that instances of a computation are always sent to different computers belonging to different participants, this approach detects result differences of all types with high probability. The probability can be made arbitrarily close to one by adjusting the number of replicas and the minimum quorum. BOINC [9] (a software platform for public-resource computing) provides built-in support for replicated computing. It allows applications to specify the parameters of replication and supports a wide range of platforms (i.e., Linux, Windows, Mac and Solaris).

Replication is not a novel technique in distributed computing. It has been used extensively for fault-

tolerance [10]. In [11] and [12], replication is used as a technique to overcome faults; in [13], dynamic model-driven replication is used to obtain high data availability; in [14], the replica scheduling of tasks is used to improve the performance of computational grids by reducing the time to solution of Monte Carlo simulations. In [15], data integrity is assessed through the use of reputation systems in fostering trust in uncertain environments (e.g., the Internet). The application of redundant result checks is combined with a reputation system.

The use of replication for detecting errors in heterogeneous computing systems, however, introduces a new difficulty: the application developer must define what it means for results to "agree". Some possibilities are:

1. Require the results to be bit-for-bit identical. This is fine for integer calculations. However, for computations that involve floating-point numbers, calculations done by different computers may differ, so that two correct results may be different.
2. The numbers making up the result must agree within application-specified tolerances. We call this "fuzzy direct comparison".
3. For some applications there is a quantity that is easily computed from the result and has the property that correct results give close values, though they may differ widely in their details. Results are considered to agree if their values of this quantity lie within a given threshold. In some cases, for example, the energy of a physical system provides such a quantity. We call this "fuzzy indirect comparison".

BOINC supports all of these policies by allowing applications to supply a function that compares sets of results.

Fuzzy comparisons, however, do not work well for some applications. For example, molecular simulations based on Monte Carlo (MC) or Molecular Dynamics (MD) are in general highly sensitive to initial conditions, and may differ depending on the machine architecture, operating system, compiler, and compiler flags. As a result, it is not easy to design an appropriate fuzzy comparison function.

In this paper we study the problem of validating results of computations that are "divergent", i.e., for which correct results can differ by arbitrary amounts. We describe two examples of divergent computations: (1) a MC simulation for protein structure assembly using the code MFold/MONSSTER [16, 17] and (2) a MD simulation for protein structure refinement using the code CHARMM [18, 19]. We look at the mathematical basis for this variability and explain it in terms of the Lyapunov exponent. We show why validation approaches based on fuzzy comparison are not applicable for these applications, and we present an alternative technique called Homogeneous Redundancy (HR). We describe its implementation in Predictor@home [1, 2], a public-resource computing project that uses the MFold/MONSSTER and CHARMM applications.

The paper is organized as follows: In Section 2 we analyze the instability in large scale molecular simulations; in Section 3 we argue that fuzzy comparison is not appropriate for these applications; in Section 4 we describe our alternative validation technique (Homogeneous Redundancy), and in Section 5 we conclude.

2 Instability in Large Scale Molecular Simulations

2.1 Lyapunov Exponent

Simulations of physical systems involve a state that evolves algorithmically over time. We define a simulation as "divergent" if two nearby states can evolve quickly into very different states. These simulations are characterized by a positive 'Lyapunov exponent' [20]. The Lyapunov exponent of a system describes the amount by which the distance between two states grows or shrinks as the systems evolve independently from those states. If for example we consider a system with a two-dimensional state space, the trajectory of two points near each other in this space can either diverge or converge. If the Lyapunov exponent is positive, the two points separate exponentially (see Figure 1.a); if the Lyapunov exponent is negative, the two trajectories converge (see Figure 1.b); and if it is zero, either the two points do not move relative to one another, or do so at some sub-exponential rate. For more complex systems with n -dimensional state spaces, a spectrum of n Lyapunov exponents needs to

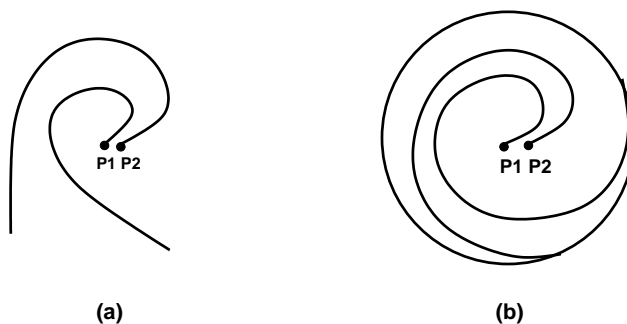


Figure 1. Lyapunov examples for a one-dimensional phase space.

be considered. The sum of all the Lyapunov exponents describes whether the state as a whole expands (positive sum) or contracts (negative sum) or is invariant (zero sum).

2.2 Performing Molecular Simulations

Molecular simulations study the structure, dynamics, thermodynamics, and other properties of molecular systems. Simulations can be performed to explore ensemble properties of molecular systems using a purely classical treatment based either on stochastic approaches such as Monte Carlo (MC) or deterministic approaches such as Molecular Dynamics (MD).

MD simulations study the motion of atoms over a total period of time that can range from some picoseconds to microseconds. A MD simulation computes the position of each atom in the system based on the force on the atom along a series of small time steps. In the MD formalism, Newton's equations are used to simulate the atomic motion of a system, generating an ensemble of molecules characterized by a set of thermodynamic state variables (e.g., temperature, pressure, and number of atoms) as well as a set of microscopic states [21]. Atom coordinates and momenta are considered in a multidimensional space of $6N$ dimensions where N is the number of atoms.

MC calculations are similar to MD calculations except that momentum degrees of freedom are not considered and no equations of motion are propagated [21]. Instead, ensembles of molecular states are generated through stochastic changes in the configuration of the molecular system.

MC and MD calculations are based on computations that are highly sensitive to initial simulation states. Both types of simulations involve iteration over large numbers of steps. The computed simulation results are usually divergent, indicating that these types of simulations are subject to positive Lyapunov exponents. Results presented in [22] support the hypothesis that such divergent behavior is due to the physical properties of the system rather than numerical instabilities in the calculations.

2.3 Experimental Cases

A molecular simulation has an initial simulation state including variables such as temperature, atom positions, etc. By performing molecular simulations with exactly the same starting state, the user might expect the same results. This is true on homogeneous systems like SMP machines or clusters, but not on public computing architectures. In this section we present two case studies based on MC and MD computations for which the results diverge despite the same initial states, because of different PC architectures.

Case Study 1: Protein structure assembly using MFold/MONSSTER

As a representative of MC simulation codes, we use the MFold/MONSSTER code for protein structure assembly. MFold/MONSSTER is based on a modeling method that uses lattice chains and MC simulations to assemble a protein structure starting from its residue side-chain input. MFold/MONSSTER is a component of Predictor@home.

To study the divergence of MFold/MONSSTER results, we consider two different structures from the set of the Sixth Biannual CASP targets¹: t0243 and t0222. Target t0243 is a small protein with 95 amino acids. Target t0222 is a larger protein with 375 amino acids. On t0243 we run 15 MC cycles (for each cycle we perform 50000 moves per residue) varying the temperature from 1.75 to 1.00 in reduced units (where $T = 1.00$ corresponds approximately to room temperature) in steps of 0.125. On t0222 we run 15 MC cycles (for each cycle we perform 13000 moves per residue) with a temperature range from 1.75 to 1.00 and step

¹CASP is the acronym of Critical Assessment of Techniques for Protein Structure Prediction

0.125.

We performed the MC simulation of each target on two different PCs with the same CPU architecture, two different operating systems, and using the same MFold/MONSSTER code compiled with two different compilers: (1) a PC with Pentium 4 processor, Linux as the OS, and the statically compiled code MFold/MONSSTER using the GNU Fortran compiler, (2) a PC with Pentium 4 processor, Windows as the OS, and the statically compiled code MFold/MONSSTER using the Intel Fortran compiler. We performed the same computation on the two machines starting each target from the same initial conditions (initial state), including the same random seed for the MC simulation.

Figure 2 shows the two final structures obtained from running the MC simulation for target t0243 on the two different PCs, where the dark blue structure is the result of the computation on the Linux machine and the light blue structure on the Windows machine. Figure 3 displays the final structures obtained from the MC simulation for the CASP target t0222. Once again the dark blue structure is the result of the simulation on the Linux machine and the light blue structure is the result of the simulation on the Windows PC. Despite the same initial conditions, the final structures are seen to be quite different.

Table 1 shows the energy values returned by MFold/MONSSTER for the two targets. For each target we observe that the energy values of the two final structures are also different but within the divergence threshold of 2.3%. The divergence threshold does not exceed the 5% threshold; however this value does not indicate whether the protein structure has been invalidated by hardware malfunctions, incorrect software modifications, or a malicious attacker.

Case Study 2: Protein refinement using CHARMM

To study the divergence of MD simulations, we consider two different CASP targets with different sizes: target t0221 and target t0235. We refine the structure of each target starting from an initial protein structure previously computed using MFold/MONSSTER with Predictor@home. The refinement protocol is based on an all-atom modeling method using the CHARMM code. This time we compare two PCs with different CPU architectures (i.e., AMD and Intel) but the same

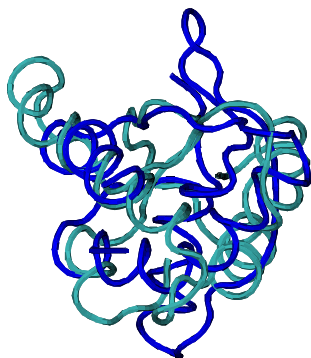


Figure 2. Final structures for CASP target t0243 composed of 165 amino acids using two PCs with different operating systems (dark blue structure -> Linux OS / GNU Fortran compiler, light blue structure -> Windows OS / Intel Fortran compiler).

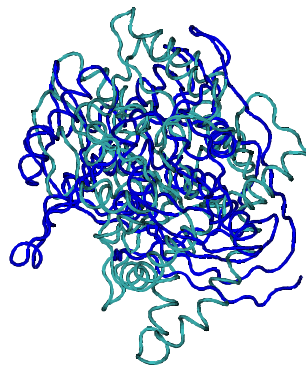


Figure 3. Final structure for CASP target t0222 composed of 375 amino acids using two PCs with different operating systems (dark blue structure -> Linux OS / GNU Fortran compiler, light blue structure -> Windows OS / Intel Fortran compiler).

Target	Num. of Amino acids	System Characteristics	Energy Values
t0222	375	GNU Fortran + Linux + Pentium 4	-3067.6093
t0222	375	Intel Fortran + Windows + Pentium 4	-3137.5281
t0243	95	GNU Fortran + Linux + Pentium 4	-826.0829
t0243	95	Intel Fortran + Widows + Pentium 4	-846.0265

Table 1. Energy values returned for the MC simulation of the two CASP targets t0243 and t0375 on two different PCs with different operating systems and compilers.

OS.

Figure 4 shows the final refined structures that result from 1000 MD steps starting from the same initial structure, initial conditions and MD random seed for CASP target t0221. The dark blue structure is the result of the simulation on a Intel processor while the light blue structure is the result of the computation on a AMD processor. Both the machines use Linux OS and the same statically linked executable. Figure 5 shows the final results for the same MD simulation for CASP target t0235. As for Figure 4, the light blue structure has been computed on a Intel processor while the light blue structure on a AMD processor. We observe that the diversity in terms of PC architecture has resulted in a significant divergence in the final structures, despite the fact that the same simulation has been replicated on both machines.

If we look at the energy values returned by the computation we can see in Table 2 that these values do not exceed the divergence threshold of 1%. However, limiting the validation only to the returned energy values does not guarantee that an ingenious malicious attacker has changed only the structure but not the energy values. Once again the divergence threshold is not a good indicator whether malicious modifications of the final structures has taken place.

3 Limitations of Fuzzy Comparison for Divergent Applications

As seen in Section 2, applications based on MD or MC can produce different outcomes for replicas of the same computation and the same starting state. The outcomes depend on the machine architecture, operating system, specific compiler and compiler flags. For

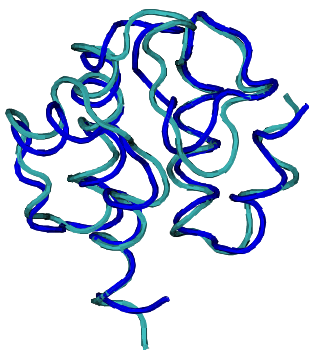


Figure 4. Final structure for CASP target t0221 composed of 95 amino acids. The two final structures have been obtained using CHARMM, starting from the same random seed but on different computer architectures (dark blue structure -> Pentium 4, light blue structure -> AMD).

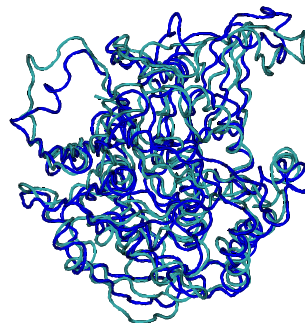


Figure 5. Final structure for CASP target t0235 composed of 499 amino acids. The two final structures have been obtained using CHARMM, starting from the same random seed but on different computer architectures (dark blue structure -> Pentium 4, light blue structure -> AMD).

Target	Num. of Amino acids	System Characteristics	Energy Values
t0221	85	GNU Fortran + Linux + Pentium 4	-5648.98
t0221	85	GNU Fortran + Linux + AMD	-5700.37
t0235	499	GNU Fortran + Linux + Pentium 4	-33894.2
t0235	499	GNU Fortran + Linux + AMD	-33911.8

Table 2. Energy values returned for the MD simulation of the two CASP targets t0221 and t0235 on two different CPU architectures.

applications that study molecular phenomena, the returned results are usually complex three-dimensional molecular structures and/or trajectories.

How might we define a "fuzzy direct comparison" function for such applications? The RMSD (Root Mean Square Deviation) is commonly used to measure the similarity of two molecular structures and can be used to quantify the divergence between two or more molecular structures from the same replicated computation. For the validation of entire trajectories, the computation and comparison of the RMSD has to be repeated for each trajectory snapshot.

An RMSD-based comparison has several drawbacks:

- For large scale molecular simulations, it would impose a large computational load the project's server complex.

- As seen in the previous sections, it is difficult or impossible to define a threshold on RMSD that distinguishes correct and incorrect results.

Alternatively, we could use fuzzy indirect comparison based on a derived quantity such as total energy. However, we have shown that it is often possible for a malicious attack to submit incorrect results that have a correct total energy. Finally, visual checking using tools like VMD [23] is not feasible for large-scale scientific simulations because of the volume of data generated.

4 Preventing Divergence on Desktop Grids

4.1 Homogeneous Redundancy

We have shown that result differences due to the heterogeneity of the computational resources make it difficult to use fuzzy comparison for result validation. In-

stead, we propose distributing the instances of a computation only among machines that are "numerically equivalent", i.e., that compute identical floating-point results. We call this policy the "homogeneous redundancy" policy (HR). Once the first instance of a task has been sent to a machine, other instances of the same task are sent only to numerically equivalent machines. Using homogeneous redundancy, it is possible to use strict equality to compare redundant results. We have implemented HR in BOINC, and used this feature in Predictor@home to successfully recognize erroneous results.

4.2 The Implementation of Homogeneous Redundancy in BOINC

How does the BOINC scheduler know when two hosts are numerically equivalent? Empirically, we have found that it is sufficient to require that the processor manufacturer (Intel, AMD, SPARC, PowerPC) and the operating system (Linux, Windows, Macintosh, Solaris) both match. This information (processor and OS) is gathered by the BOINC client and made available to the scheduler. There are generally a sufficient number of hosts of each "equivalence class" to provide the needed redundancy. We use a field of the task table in the database to store a code representing the equivalence class of the host to which an instance of a task (replica) has been distributed, if any. This approach allows us to keep the scheduling workload low due to the HR on the server and the database: the server is not required to additionally access the database to retrieve information about participating hosts.

4.3 Validation Results

We have collected the results related to the validation of replicated task computations on Predictor@home for the two applications MFold/MONSSTER and CHARMM over two overlapping intervals of time: from August 17 to August 30 for MFold/MONSSTER and August 26 to August 30 for CHARMM. The several tasks have been distributed using the same server over a farm of about 12,000 heterogeneous PCs connected to the Internet. Since the computation of MFold/MONSSTER tasks is on average three times longer than the computation of CHARMM tasks, we collected data for the first application over a longer in-

terval of time.

Table 3 presents the number of results that returned to the server on August 30 for the two applications and have been validated. Each task has been replicated and distributed to the clients at least three times. The final validation on the server takes place when at least two of three results for a given task has been returned.

Results	MFold	CHARMM
total	402,002	138,591
valid	380,269 (94.6%)	99,352 (71.7%)
invalid	11,690 (2.9%)	12,139 (8.8%)
error	10,043 (2.5%)	27,100 (19.5%)

Table 3. Validated results observed on Predictor for the two applications MFold/MONSSTER and CHARMM over two overlapping intervals of time.

In the table, we classify as *valid* those results that return to the server after being successfully completed and pass the HR strict equality comparison; that means at least another replicated result of the same task has been returned and the two computed molecular structures are identical. If the result computation has been successfully completed on a client (no error has been reported by the client during the downloading, execution or uploading phase) but the result fails to pass the HR strict equality comparison, it is classified as an *invalid* result. All the results that have encountered any problem during their execution on the client (during the downloading, execution or uploading phase) are classified as *error* results. If needed, BOINC proceeds to generate new replications for those tasks that are classified as error or invalid.

CHARMM is characterized by a higher number of floating point operations per second than MFold/MONSSTER. This makes CHARMM more vulnerable to crashes and hardware errors during floating point calculations, explaining in part the higher ratio of invalid results and errors for this application. However, such a high percentage of invalid results for CHARMM (8.8%) are unlikely to be only due to CPU overclocking. We are currently looking at other fac-

tors that could be a cause for such a high percentage of invalid results for CHARMM.

5 Conclusion

Simulations using public resource computing requires validation of the returned results to ensure that they are not affected by hardware errors, incorrect software modifications, or malicious attack. Replication is a useful validation method. However, for molecular simulations, replica results might significantly diverge because of differences in host hardware and software. Homogeneous Redundancy (HR) is based on the distribution of replicas among numerically equivalent machines. By using HR, we can instantaneously recognize erroneous results. We have presented the HR policy and shown how it has been implemented within the BOINC framework to support result validation in Predictor@home, a world-community effort to predict protein structure from sequence. HR allows us to apply a straightforward validation based on strict equality to compare redundant results.

Acknowledgments

Financial support from the National Institutes of Health Grant, RR12255, and from the National Science Foundation Grant SCI/0221529 is greatly appreciated. Financial support through the NSF sponsored Center for Theoretical Biological Physics (grants# PHY-0216576 and 0225630) and the LJIS Fund are acknowledged. We wish to thank Rom Walton and Andre Kerstens for their invaluable help with Predictor@home, Chahm An for designing the input files for MFold and CHARMM simulations, and the Predictor@home and BOINC community for donating their computing cycles.

References

- [1] M. Taufer, C. An, A. Kerstens, and C.L. Brooks III. Predictor@home: A "Protein Structure Prediction Supercomputer" Based on Public-Resource Computing. In *Proceedings of the Fourth IEEE International Workshop on High Performance Computational Biology (HiCOMB'05)*, Denver, CO, April 2005.
- [2] C. An, M. Taufer, and C.L. Brooks III. Predictor@home: A Multiscale, Distributed Approach for Protein Structure Prediction. In *Proceedings of the 6th Community Wide Experiment on the Critical Assessment of Techniques for Protein Structure Prediction (CASP6)*, Gaeta, Italy, December 2004.
- [3] M. Shirts and V.S. Pande. Screen Savers of the World, Unite! *Science*, 2000.
- [4] S.M. Larson, C.D. Snow, M. Shirts, and V.S. Pande. Folding@home and Genome@home: Using Distributed Computing to Tackle Previously Intractable Problems in Computational Biology. *Computational Genomics*, 2002. Richard Grant, editor, Horizon Press.
- [5] FightAIDS@home Project Page. <http://fightaidsathome.scripps.edu/>.
- [6] distributed.net Project Page. <http://distributed.net/>.
- [7] G. Tarsy and N. Toda. Floating-Point Computing: A Comedy of Errors? Sun Microsystems, 2004.
- [8] G. Bosilca, F. Cappello, A. Djilali, G. Fedak, T. Her-aet, and F. Magniette. Performance Evaluation of Sandboxing Techniques for P2P and Global Operating Systems. 2002.
- [9] D. Anderson. BOINC: A System for Public-Resource Computing and Storage. In *Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04)*, Pittsburgh, PA, November 2004.
- [10] D. McEvoy. The Architecture of Tandem's NonStop System. In *Proceedings of the ACM '81 Conference*, page 245. ACM Press, 1981.
- [11] C. Aktouf, O. Benkahla, C. Robach, and A. Guran. Basic Concepts and Advances in Fault-Tolerant Computing Design. *World Scientific Publishing Company*, 1998.
- [12] L.F.G. Sarmenta. Sabotage-Tolerance Mechanisms for Volunteer Computing Systems. In *Proceedings of ACM/IEEE International Symposium on Cluster Computing and the Grid (CCGrid'01)*, Brisbane, Australia, May 2001.
- [13] A. Iamnitchi and I. Foster. Improving Data Availability through Dynamic Model-Driven Replication in Large Peer-to-Peer Communities. In *Proceedings of ACM/IEEE International Symposium on Cluster Computing and the Grid (CCGrid'02)*, Munich, Germany, May 2002.
- [14] Y. Li and M. Mascagni. Improving Performance via Computational Replication on a Large-Scale Computational Grid. In *Proceedings of ACM/IEEE International Symposium on Cluster Computing and the Grid (CCGRID'03)*, Tokyo, Japan, May 2003.
- [15] A. Gilbert, A. Abraham, and M. Paprzycki. System for Ensuring Data Integrity in Grid Environments International. In *Conference on Information Technology: Coding and Computing (ITCC'04)*, Las Vegas, Nevada, April 2004.

- [16] A. Kolinski and J. Skolnick. Assembly of Protein Structure from Sparse Experimental Data: An Efficient Monte Carlo Model. *Proteins: Structure, Function, and Genetics*, 32:475–494, 1998.
- [17] J. Skolnick, A. Kolinski, and A.R. Ortiz. MONSSTER: A Method for Folding Globular Proteins with a Small Number of Distance Restraints. *J. Mol. Biol.*, 265:217–241, 1997.
- [18] B. R. Brooks, R. E. Bruccoleri, B. D. Olafson, D. J. States, S. Swaminathan, and M. Karplus. Charmm: A Program for Macromolecular Energy Minimization, and Dynamics Calculations. *J. Comput. Chem.*, 4:187–217, 1983.
- [19] A.D. MacKerell Jr., B. Brooks, C.L. Brooks III, L. Nilsson, B. Roux, Y. Won, and M. Karplus. CHARMM: The Energy Function and Its Parameterization with an Overview of the Program. *The Encyclopedia of Computational Chemistry*, 1:271–277, 1998. P. v. R. Schleyer et al., editors (John Wiley and Sons: Chichester).
- [20] A.M. Ozorio de Almeida. *Hamiltonian Systems: Chaos and Quantization*. Cambridge University Press, Cambridge, UK, 1988.
- [21] A. Leach. *Molecular Modelling/2.Edition*. Prentice Hall, 2001.
- [22] M. Braxenthaler, R. Ron Unger, D. Auerbach, J.A. Given, and J. Moul. Chaos in Protein Dynamics. *Proteins*, 29(417–425), 1997.
- [23] W. Humphrey, A. Dalke, and K. Schulten. VMD - Visual Molecular Dynamics. *J. Molec. Graphics*, 14(1):33–38, 1996.

Biographies

Michela Taufer is Assistant Professor in Computer Science at the University of Texas in El Paso. Her research interests include new algorithms and architectures for resource- and time-expensive applications in computational chemistry, physics, and biology; effective migration of scientific computation codes to novel, state-of-the-art grid platforms; and performance analysis, -modeling and -optimization of applications on distributed systems.

Dr. Taufer earned her MS from the University of Padova and her Ph.D. from the Swiss Federal Institute of Technology (ETH), both in Computer Science. She has post-doc experience at UC San Diego, the San Diego Supercomputer Center, and The Scripps Research Institute (TSRI).

David P. Anderson received graduate degrees in Mathematics and Computer Science at the University of Wisconsin. From 1985 to 1992 he served on the faculty of the U.C.

Berkeley Computer Science Department. The topics of his published research include distributed operating systems, realtime and multimedia systems, graphics, and computer music.

Dr. Anderson is currently a Research Scientist at the U.C. Berkeley Space Sciences Laboratory, where as director of SETI@home and BOINC projects he conducts research in large-scale public distributed computing.

Pietro Cicotti is a graduate student at the University of California, San Diego (UCSD), member of the Scientific Computation Group. His research is in the area of Parallel Scientific Computing and the investigation of techniques aimed at reducing the development time of high-performance scientific applications, focusing on the applications needs to improve running time. His previous work involved grids and desktop grids; in 2004 he had a developer position at the San Diego Supercomputer Center (SDSC) and at The Scripps Research Institute (TSRI). He received his Laurea in Computer Science with emphasis on distributed systems at the University of Bologna, Italy, in 2003.

Charles L. Brooks III is Professor of Molecular Biology at The Scripps Research Institute. Prior to moving to TSRI in 1994, Professor Brooks was professor of Chemistry at Carnegie Mellon University, where he began as an assistant professor in 1985. He received his undergraduate training in chemistry, physics and mathematics from Alma College. He studied chemical physics at Purdue University, where he received his Ph.D. in 1982, and was a postdoctoral fellow at Harvard University from 1982-85.

Professor Brooks' research focuses on the applications of statistical mechanics, quantum chemistry, and computational methods to solve chemically and physically oriented problems in biology. Among the more recent honors and fellowships earned by Professor Brooks were A. P. Sloan Foundation Research Fellow, 1990-1993; Visiting Scholar Grant from Swedish National Research Council, June-August, 1992; Computer world Smithsonian Institute Award, 1997 and American Association for the Advancement of Science Fellow, 2001.