Study of Computing Consolidation Techniques in Computational Protein Loop Structure Modeling

Yaohang Li and Douglas Wardell
Department of Computer Science
North Carolina A&T State University
{yaohang, dwardell}@ncat.edu

Keywords: Protein Modeling, Consolidation

Abstract

In this article, we advocate the approach of "computing consolidation" to achieve efficient usage of computer resources in protein structural modeling applications. The fundamental idea of computing consolidation is to increase computational density to a processor in order to increase CPU and other resources utilization rate. We use our computational protein loop structure modeling application with clear memory-intensive and computation-intensive energy function evaluation components as an example to investigate the effectiveness of computing consolidation. We study various implementations of computing consolidation, including process consolidation, fine-grain and coarse-grain thread consolidations on a variety of hardware architectures. Our computational results show that thread consolidation leads to more significant speedup due to the cache effect and the lower cost in thread management than process management. Moreover, different thread consolidation implementations are suitable for different hardware architectures. By consolidating threads carrying out energy function components with heterogeneous resource demands, energy-function-level thread consolidation outperforms iteration-level thread consolidation with over 10% speed improvement in single-CPU processors. However, on hyper-threading-enabled processors or multi-core processors, iteration-level thread consolidation enables more computation overlaps in the concurrent threads carrying out independent iterations, which yields more aggressive computation speed improvement.

1. Introduction

Accurately modeling protein structure or complex is considered one of the most significant grand challenges that have broad economic and scientific impact. Due to the large protein conformation space as well as the complication of the energy function describing the interaction among protein atoms, many protein modeling applications are rather computational costly, which demand large amount of computational resources and time. The typical computation time for the protein structure modeling applications ranges from several hours to several days or even longer. In practice, the protein modeling programs may be required to be carried out on many protein targets. For example, in system biology study, the structure models and interactions on thousands of bacterial proteins are needed to understand the function of a biological system. Assuming that the average protein modeling computation time is around 1 hour, systematic study of a biological system with ~10,000 proteins can easily reach 10,000 hours of computation time. Such heavy computations are usually carried out on a large-scale computation platform such as a Tera-scale high-performance supercomputer cluster or a computational grid with large number of distributed computational resources [1]. The computation time reduction of a protein modeling program, even only a small fraction, can free up considerably large amount of computational resources.

Parallel computing is the most common approach to reduce the computation time of the protein modeling applications. By scattering the computation to parallel processors, the overall application computation time may be reduced. However, this approach demands significantly more computational resources, which is not suitable in system biology computation when modeling a lot of proteins or protein complexes are required. For large number of protein simulations, the popular approach is to execute the protein modeling application in an embarrassingly parallel mode on large-scale computational platforms, such as folding@home [18], rosetta@home [19], and predictor@home [20].

Our study shows that many current computational protein structure modeling programs have relatively low CPU utilization rate or relatively large percentage of wasted CPU cycles. Figure 1 shows the CPU utilization rate in several popular protein structure modeling programs, including Rosetta *ab initio*[1] (*ab initio* protein folding) [2, 3], MODELLER[2] (comparative modeling) [4], GROMACS[3] (protein molecular dynamics) [5], and GEOFOLD[4] (protein unfolding simulation) [6] as well as our Protein Loop Structure Modeling program[5] [7]. These computational experiments are carried out exclusively on a single-processor computer with a 1.6GHz Intel Pentium 4 CPU, 512MB memory, 8K L1 cache, and 256K L2 cache. The application performance data is obtained by PAPI [8]

[1] Computation on target 1cg5 B chain using Rosetta default parameters.
[2] Computation on target 1fdx using MODELLER default parameters.
[3] Computation using gmxdemo with GROMACS default parameters.
[4] Computation on target 1lz1 using GEOFOLD default parameters.
[5] Computation on target 1cyo(32:43) using default parameters.

and PerfSuite [9], which are performance tuning software packages based on hardware performance counters. One can notice that these applications either have relatively low CPU utilization rate or have relatively large portion of the CPU cycles stalled on various resources due to cache missing, branch-misprediction, or data dependency. This is due to the fact that these protein structure modeling programs usually involve large number of operations in file system I/O, database queries, allocating and accessing of large-scale memory, or network communication, which causes low CPU utilization rate or stalled CPU cycles on various resources. Improving the resource utilization rate will lead to performance improvement in these protein structure modeling applications.

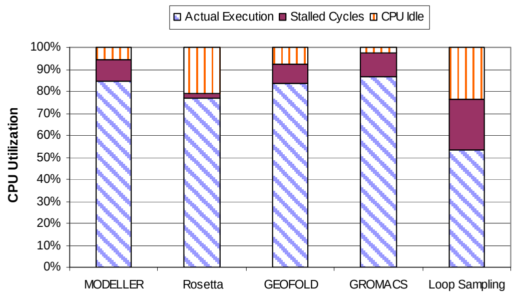


Figure 1: CPU Utilization in Several Popular Protein Modeling Applications as well as the Protein Loop Structure Modeling Program

In this article, we address the resource inefficiency problem presented in computational protein modeling applications via the "computing consolidation" techniques. The fundamental idea of consolidation is to increase computing density to a processor in order to improve the resource utilization rate and thereby speed up the computing task or improve the overall system throughput. Although computing consolidation technique intends to improve resource utilization efficiency, blind computing consolidation may lead to performance degrading. For example, consolidating multiple memory-intensive threads may cause interferences in cache or Translation Lookaside Buffers (TLB) and thus slow down these threads. We hereby investigate various implementations of computing consolidation, including process consolidation, coarse-grain thread consolidation, and fine-grain thread consolidation. We use our protein loop structure modeling application [7] as an example program to study the effectiveness of various implementations of computing consolidation on a variety of hardware architectures, including single CPU processors, hyper-threading-enabled processors, and multi-core processors.

2. Consolidation

"Consolidation" is an important approach to achieve the efficient usage of computer resources. The most well-known consolidation approach is "server consolidation" [10] by packing applications onto fewer servers so as to improve server usage efficiency and thus reduce the total number of servers required in an organization. In this article, we advocate the approach of consolidating concurrent processes or threads to improve resource utilization efficiency so as to speed up the computational protein structure modeling applications. The rationale of consolidation is to increase computing density to a processor or server in order to improve the CPU and other resources utilization.

To increase computing density, multiple concurrent, independent processes can be generated and assigned to a computer. For example, processes of simulation on different protein targets can be carried out simultaneously on a computer. This approach is referred to as process consolidation. In process consolidation, when one process is waiting for resources, the others may be swapped in and take advantage of the idle CPU. The major overhead in process consolidation is the cost of process switching, which is particularly costly when a process occupies a large memory space.

Thread consolidation is an alternative way to increase computing density to a computer. In thread consolidation, concurrent threads are spawned to carry out parallel components in the application. Similar to process consolidation, when one thread is waiting for an external event to happen, the other active threads can be swapped in the execution mode. As a result, the computational resource utilization rate can be improved and thus the protein energy function evaluation time can be reduced. Thread consolidation will be particularly effective if the computational components carried out by concurrent threads have different demands of computational resources. The overhead introduced by thread consolidation is thread management, including operations of thread creation, scheduling, and termination.

Thread consolidation requires modification of the original program to support multithreads. In the protein structure modeling programs, one has to identify parallel components and then implement thread functions to carry out these components. Compared to thread consolidation, the key advantage of process consolidation is that there is minor or no requirement to modify the original program. However, thread consolidation may lead to more aggressive performance improvement. Because multiple threads share the same program space, when several threads work on the same set of data, they can actually share caching, which will lead better cache usage. Moreover, if a thread is stalled due to cache misses, branch misprediction, or data dependency, the other threads may take advantage of the unused execution resources. Figure 2 shows the conceptual illustrations of process consolidation and thread consolidation.

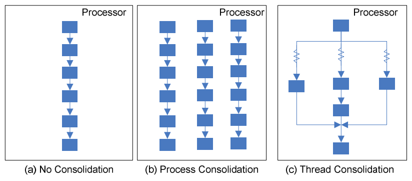


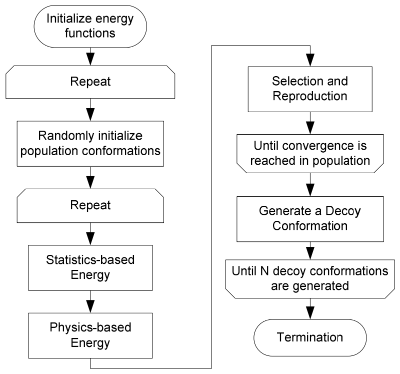
Figure 2: Conceptual Illustrations of Process and Thread Consolidation

3. Consolidation Implementation in Protein Loop Structure Modeling Program

3.1. *ab initio* Protein Loop Structure Sampling

	Statistics-based Energy	Physics-based Energy
Total Cycles	11,955,360,744	7,711,137,805
Floating-point operations	1,029,555,315	1,297,209,229
Floating-point operations/cycle	0.086	0.168
L2 cache access	45,896,473	38,931,319
L2 cache misses	18,485,848	392,170
% L2 cache misses	40.28%	1.01%
% cycles stalled on resources	16.4%	0.3%

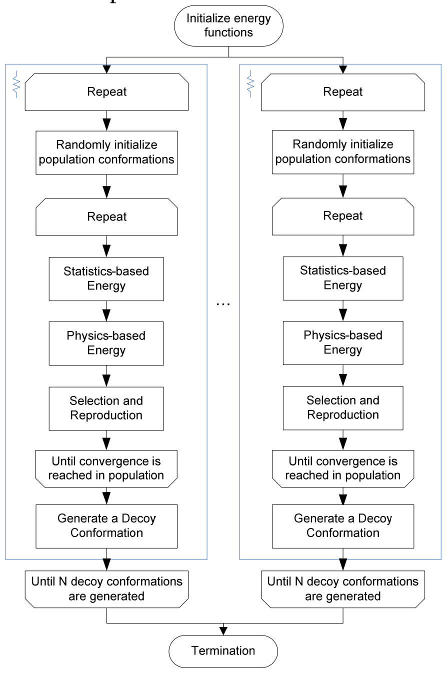
Table 1: Analysis of Performance and Resource Demands in Evaluations of Distance-based Energy, Torsion-based Energy, and Soft-Sphere Energy in the Protein Loop Structure Sampling Program


Initialize energy functions
Repeat
Randomly initialize population conformations
Repeat
Statistics-based Energy
Physics-based Energy
Selection and Reproduction
Until convergence is reached in population
Generate a Decoy Conformation
Until N decoy conformations are generated
Termination
Figure 3: Protein Loop Structure Modeling Flowchart

The protein loop structure modeling program is rather computational costly, where generation of 1,000 decoy conformations of a moderate size protein loop typically requires over a day's computation on a single processor. The most time-consuming part is the evaluation of the multiple energy functions, which occupies more than 95% of the overall computational time. Our further analysis shows that the computational evaluations of the energy functions used in the protein loop structure modeling program exhibit rather different computational resources requirements. Table 1 shows the performance and resource demands in evaluations of statistics- and physics-based energy functions in the protein loop structure sampling program. The performance data is obtained on a single processor computer with 1.6GHz Intel Pentium 4 CPU, 512MB memory, 8K L1 cache, and 256K L2 cache and the application performance data are obtained by PAPI and PerfSuite. One can find that the physics-based energy function is rather computation-intensive, with high density of floating point operations, low cache missing rate, and low percentage of stalled cycles. In contrast, the statistics-based energy function is memory-intensive with high percentage of L2 cache misses, which leads to high percentage of stalled CPU cycles. This is due to the fact that the physics-based energy functions rely on the intensive calculation of energy while the statistics-based energy functions derive their potential from statistical observations, which usually requires searching a large data set loaded in memory [16]. Computing consolidation techniques may be used to take advantage of the heterogeneous demands of resources in the multiple energy functions and achieve performance improvement by enhancing resources utilization.

3.2 Consolidation Implementation

Compared to thread consolidation, the implementation of process consolidation is trivial, where multiple processes can run in parallel with different parameters. We implemented two versions of thread consolidation, including the coarse-grain thread consolidation at iteration level and the fine-grain thread consolidation at energy function evaluation level. The coarse-grain thread consolidation spawns concurrent threads to carry out independent iterations of decoy production. The fine-grain thread consolidations take place at the phase of multiple energy functions evaluation, where each thread carries out the computation of statistics- or physics-based energy functions [21]. Figures 4 and 5 illustrate the coarse-grain thread consolidation at iteration level and the fine-grain thread consolidation at energy function evaluation level, respectively. The energy-function-level thread consolidation allows direct consolidation of memory-intensive and computation-intensive terms and the iteration-level thread consolidation allows more computation overlaps in the concurrent threads.


Initialize energy functions
Repeat
Randomly initialize population conformations
Repeat
Statistics-based Energy
Physics-based Energy
Selection and Reproduction
Until convergence is reached in population
Generate a Decoy Conformation
Until N decoy conformations are generated
...
Termination
Figure 4: Coarse-Grain Thread Consolidation at Iteration Level

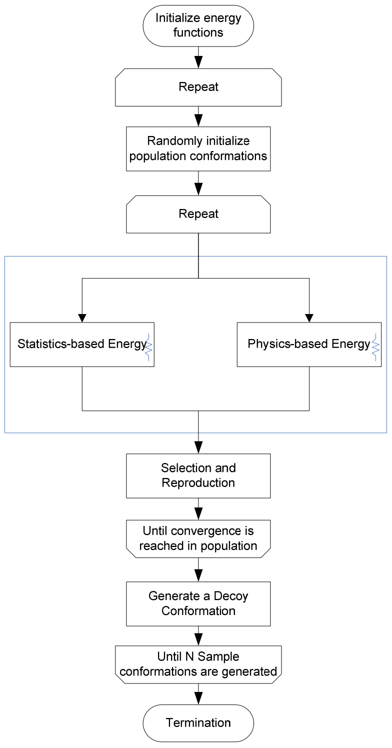

Initialize energy functions
Repeat
Randomly initialize population conformations
Repeat
Statistics-based Energy
Physics-based Energy
Selection and Reproduction
Until convergence is reached in population
Generate a Decoy Conformation
Until N Sample conformations are generated
Termination

Figure 5: Fine-Grain Thread Consolidation at Energy Function Level

4. Computational Results

We benchmarked the protein loop structure sampling programs using various consolidation implementations as well as the serial implementation on various computing systems with single CPU, hyperthread CPU, or multi-core processors. All executables are generated by the GCC compiler with "–O3" optimization option. The thread consolidation programs use the pthread library. All computations are executed exclusively on a targeted computer system by modeling a 12-residue loop (181:192) in protein 1akz to produce 10 decoys. The computation time measures are based on the average of 10 repeated runs. To ensure that the serial, process consolidation, and thread consolidation programs produce exactly the same simulation results in generating each decoy, the Scalable Parallel Random Number Generators (SPRNG) library [17] is employed to produce reproducible parallel pseudorandom number streams so that the same pseudorandom number sequence is used in the serial as well as the consolidation programs when the same decoy conformation is generated.

4.1. Consolidation on Single-CPU Processors

Figures 6 and 7 show the decoy generation time in process consolidation, iteration-level thread consolidation, and energy-function-level thread consolidation on a computer system with a single-CPU Intel Pentium 4 processor and a system with a PowerPC processor, respectively. Table 2 lists the percentage of speed improvement in various consolidation implementations on single-CPU processors. It is interesting to find that both the process consolidation and the thread consolidation implementations lead to certain performance improvement; however, the speed improvements are more significant in the thread consolidation implementations. This is due to less cost in thread management than process management and, possibly more importantly, the cache effect, where concurrent threads may lead to more efficient use of cache. Moreover, the fine-grain thread consolidation at energy function level yields more aggressive speedup than coarse-grain thread consolidation at iteration level, which leads to over 10% speed improvement. This is because the fine-grain implementation consolidates computation terms at energy function level with clear heterogeneous resource demands while in coarse-grain thread consolidation, computations carried out by concurrent threads at iteration level may occasionally demand the same resources, which will lead to resource contention or cache interference.

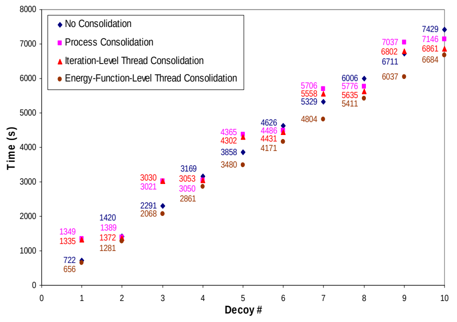


Figure 6: Decoy Generation Time in Various Consolidation Implementations on a Computer System with a single CPU processor with 1.6GHz Intel Pentium 4 CPU, 512MB memory, and 256K L2 cache

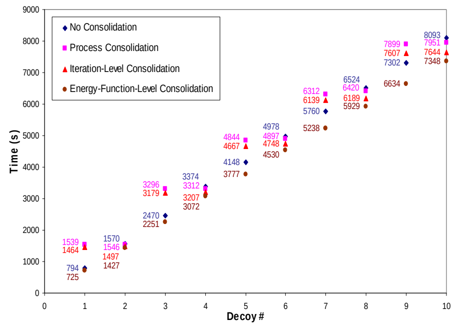


Figure 7: Decoy Generation Time in Various Consolidation Implementations on a Computer System with 1.67GHz PowerPC CPU, 1GB memory, and 512KB L2 cache

	Pentium 4	PowerPC
Process Consolidation	3.96%	1.79%
Iteration-level Thread Consolidation	8.27%	5.87%
Energy-Function-Level Thread Consolidation	11.15%	10.14%

Table 2: Speed Improvement Comparison in Various Consolidation Implementations on Traditional Single-CPU Processors

4.2. Consolidation on Hyper-Threading-Enabled and Multi-Core Processors

Figure 8 shows the decoy generation time in various consolidation implementations on a hyper-threading-enabled processor. The hyper-threading technology allows concurrent execution of multiple processes or threads. When the execution resources are not used by one process or thread, the other process or thread can take advantage of these execution resources. As a result, consolidation implementations lead to more significant performance improvements than that on traditional single-CPU processors, as shown in Table 3. Similar to that of single-CPU processors, the thread consolidation implementations yield more significant speed improvement than the process consolidation implementation on the hyper-threading-enabled processor. Due to more computation overlaps in the concurrent threads, the iteration-level thread consolidation can take advantage of the hyper-threading facility more efficiently than energy-function-level consolidation and thus lead to slightly more speed improvement.

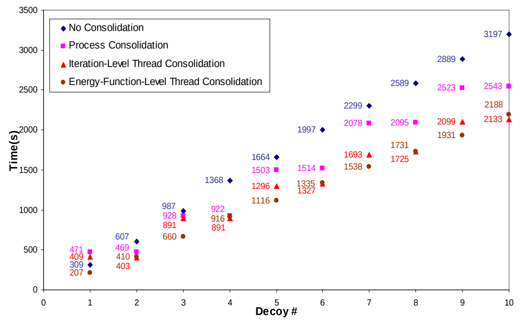


Figure 8: Decoy Generation Time Comparison of Consolidation Implementations on a Computer System with a 3.40GHz Intel Pentium 4 Hyper-threading-enabled CPU, 1GB memory, and 1MB L2 cache

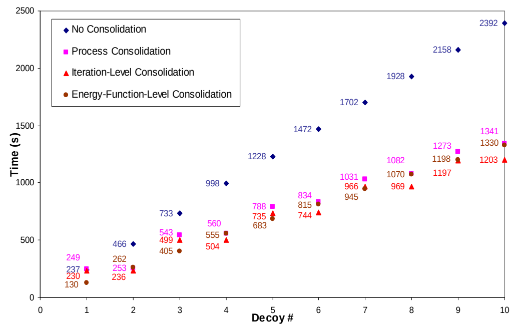


Figure 9: Decoy Generation Time Comparison of Consolidation Implementations on a Computer System with an Intel Core2 Duo 2.33GHz Processor, 2GB memory, and 4MB L2 cache

	Hyper-threading	Dual-core
Process Consolidation	25.72%	78.37%
Iteration-level Thread Consolidation	49.88%	98.84%
Energy-Function-Level Thread Consolidation	46.12%	79.85%

Table 3: Speed Improvement Comparison in Various Consolidation Implementations on a Hyper-threading-enabled Processor and a Dual-core Processor

Figure 9 depicts the decoy generation time in various consolidation implementations on a dual-core processor. It is well-known that a dual-core processor may potentially lead to 100% speed improvement for computation-intensive applications. However, for memory-intensive applications, typical speed improvement is 70% or less [22]. This is due to the fact that memory bandwidth becomes a critical resource in multi-core architectures – extra thread running on an extra core will post extra traffic on the memory bus. As shown in Table 3, with sufficient computation overlaps in the concurrent threads, the iteration-level thread consolidation implementation in the protein loop structure modeling program with hybrid memory-intensive and computation-intensive terms yields near ideal speed improvement (98.84%). On the other hand, because evaluating statistics-based energy functions requires more computation time than physics-based energy functions in the protein loop modeling program, the energy-function-level thread consolidation implementation leads to unbalanced workload on the cores, which yields less significant speedup than the iteration-level thread consolidation. It is important to notice that dividing the statistics-based energy computation into more concurrent threads may help balance the workload on the cores, but actually lead to worse performance (only 68.22% speed improvement in our computational experiment when 2 threads are spawned to carry out statistics-based energy function evaluations). This is due to the fact that running multiple memory-intensive threads on both cores can lead to severe contention to the memory bus.

4.3. Configuring Thread Consolidation

An important question in thread consolidation is what the optimal number of concurrent threads is. Figure 10 illustrates the trend of speed improvement as the number of threads increases in the iteration-level thread consolidation on computer systems with single-CPU, hyper-threading, and dual-core processors. One can find that using 2 or 3 threads to carry out concurrent iterations yields the optimal speed improvement in the protein loop structure modeling program. When more threads are spawned, the thread consolidation performance starts to decrease due to increase of thread management overhead as well as potential resource contention among these concurrent threads.

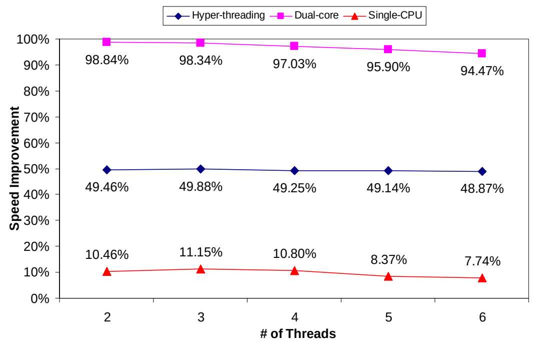


Figure 10: Impacts of Number of Threads on Speed Improvement in Iteration-level Thread Consolidation on Single-CPU, Hyper-threading-enabled, and Dual-Core Processors

5. Conclusions and Future Research Directions

In this article, we use our computational protein loop structure modeling application as an example to study the effectiveness of various computing consolidation strategies on a variety of hardware architectures. Our computational results in this example shows that thread consolidation can lead to more significant performance improvement than process consolidation due to the cache effect as well as the lower cost in thread management than process management. Moreover, the fine-grain thread consolidation implementation at energy function level leads to more aggressive speedup than the coarse-grain thread consolidation implementation at iteration level on traditional single-CPU processors because of directly consolidating threads carrying out computation-intensive and memory-intensive terms. On the other hand, with more computation overlaps in concurrent threads, the iteration-level thread consolidation takes advantage of multi-thread support hardware presented in the hyper-threading-enabled and multi-core processors more efficiently and thus yields to more aggressive speed improvement than energy-function-level consolidation.

The computing consolidation technique is particularly suitable for many computational biology applications with heterogeneous resources requirements. In our future research, we plan to apply the computing consolidation technique to several important protein modeling applications, including *ab initio* protein folding, protein-ligand docking, and protein-protein interaction, with expectation to achieve resource efficiency and performance improvement. We also plan to develop an intelligent thread consolidation scheme, which can adaptively adjust the thread consolidation configuration in different architectures to achieve optimal consolidation speedup.

Acknowledgements

The work is partially supported by NSF under grant numbers CCF-0829382 and CCF-0845702.

References

[1] I. Foster, C. Kesselman, S. Tueske, "The Anatomy of the Grid," *Int. J. of Supercomputer Applications*, **15**(3), 2001.

[2] Bonneau, J. Tsai, I. Ruczinski, D. Chivian, C. Rohl, C. E. Strauss, D. Baker, "Rosetta in CASP4: progress in ab initio protein structure prediction," *Protein*, **5**: 119-126, 2001.

[3] P. Bradley, D. Chivian, J. Meiler, K. M. Misura, C. Rohl, W. Schief, W. J. Wedemeyer, O. Schueler-Furman, P. Murphy, J. Schonbrun, C. E. Strauss, D. Baker, "Rosetta predictions in CASP5: Successes, failures, and prospects for complete automation." *Proteins*, **53**: 457-468, 2003.

[4] M. A. Marti-Renom, A. Stuart, A. Fiser, R. Sánchez, F. Melo, A. Sali, "Comparative protein structure modeling of genes and genomes," *Annu. Rev. Biophys. Biomol. Struct.*, **29**, 291-325, 2000.

[5] B. Zagrovic, C. D. Snow, M. R. Shirts, V. S. Pande, "Simulation of Folding of a Small Alpha-helical Protein in Atomistic Detail using Worldwide Distributed Computing," *Journal of Molecular Biology*, **323**(5): 927-937, 2002.

[6] V. Ramakrishnan, S. M. Salem, M. J. Zaki, S. J. Mathews, S. Srinivasan, W. Colon, C. Bystroff, "Developing a detailed mechanistic model for protein unfolding," in preparation, 2008.

[7] Y. Li, I. Rata, E. Jakobsson, "Multi-Scoring Functions Sampling in Protein Loop Structure Prediction," *submitted to Journal of Computational Biology*, 2008.

[8] S. Browne, C. Deane, G. Ho, P. Mucci, "PAPI: A Portable Interface to Hardware Performance Counters," *Proc. of Department of Defense HPCMP Users Group Conf.*, 1999.

[9] R. Kufrin, "PerfSuite: An Accessible, Open Source Performance Analysis Environment for Linux," *6th Int. Conf. on Linux Clusters: The HPC Revolution*, 2005.

[10] M. R. Marty, M. D. Hill, "Virtual hierarchies to support server consolidation," *Proc. of 34th Annual International Symposium on Computer Architecture*, 2007.

[11] K. Deb, "Multi-Objective Optimization using Evolutionary Algorithms," *Wiley-Interscience Series in Systems and Optimization*, 2001.

[12] A. Rojnuckarin, S. Subramaniam, "Knowledge-based interaction potentials for proteins", *Proteins*, **36**: 54-67, 1999.

[13] I. Rata, Y. Li, E. Jakobsson, "Backbone Statistical Potential from Local Sequence-Structure Interactions in Protein Loops," Journal of Physical Chemistry B, in press, 2010.

[14] H. Zhang, L. Lai, Y. Han, Y. Tang, "A Fast and Efficient Program for Modeling Protein Loops," *Biopolymers*, **41**: 61-72, 1997.

[15] R. Storn, K. Price, "Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces," *J. of Global Optimization*, **11**(4):341–359, 1997.

[16] G. Helles, "A Comparative Study of the Reported Performance of *ab initio* Protein Structure Prediction Algorithms," *J. R. Soc. Interface*, **5**(21): 387-396, 2008.

[17] M. Mascagni, A. Srinivasan, "Algorithm 806: SPRNG: A Scalable Library for Pseudorandom Number Generation," *ACM Trans. on Mathematical Software*, **26**: 436-461, 2000.

[18] B. Zagrovic, C. D. Snow, M. R. Shirts, V. S. Pande, "Simulation of Folding of a Small Alpha-helical Protein in Atomistic Detail using Worldwide Distributed Computing," *Journal of Molecular Biology*, **323**(5): 927-937, 2002.

[19] R. Das, B. Qian, S. Raman, R. Vernon, J. Thmpson, P. Bradley, S. Khare, M. D. Tyka, D. Bhat, D. Chivian, D. E. Kim, W. H. Sheffler, L. Malmstrom, A. W. Wollacott, C. Wang, I. Andre, D. Baker, "Structure prediction for CASP7 targets using extensive all-atom refinement with Rosetta@home," *Protein*, **69**(8)**:** 118-128, 2007.

[20] M. Taufer, C. An, A. Kerstens, C. L. Brooks, "Predictor@Home: A Protein Structure Prediction Supercomputer Based on Public-Resource Computing," *Proc. of 4th IEEE International Workshop on High Performance Computational Biology (HiCOMB)*, 2005.

[21] Y. Li, D. Wardell, V. Freeh, "A Resource-Efficient Computing Paradigm for Computational Protein Modeling Applications," *Proc. of the 8th International Workshop on High Performance Computational Biology (HiCOMB)*, 2009.

[22] J. Carter, Y. He, J. Shalf, H. Shan, E. Strohmaier, H. Wasserman, "The Performance Effect of Multi-core on Scientific Applications," *Proc. of Cray User Group Conference*, 2007.