Database/Software Paper

Free Energy Landscape Analysis System Based on Parallel Molecular Dynamics Simulation

Masakazu Sekijima,[†1] Jun Doi,[†2] Shinya Honda,[†3]
Tamotsu Noguchi,[†1] Shigenori Shimizu[†2]
and Yutaka Akiyama[†1,†4]

We created a Free Energy Landscape Analysis System based on a parallelized molecular dynamics (MD) simulation adapted for the IBM Blue Gene/L supercomputer. We begin with an outline of our Free Energy Landscape Analysis system. Next we discuss how Parallel MD was tuned for Blue Gene/L. We then show the results for some test targets run on Blue Gene/L, including their efficiency. Finally, we mention some future directions for extension of this project.

1. Introduction

Molecular structures and functions have relationships with each other. However it is not easy to predict molecular functions from the rigid structures recorded in the Protein Data Bank (PDB) [1] files, because biomolecules are fluctuating in living cells. A quantitative understanding of the relationships between structure, dynamics, stability, and functional behaviors of proteins are of paramount importance. There are essential themes to represent fluctuations throughout computational biology [2].

In this work, we used molecular dynamics (MD) simulation to evaluate molecular fluctuations. MD simulations are widely used for simulating the motions of molecules. Rapidly increasing computational power has made MD simulation a powerful tool for studying the structure and dynamics of biologically important molecules [3]. To understand the thermodynamics and kinetics of protein folding, we developed a free energy landscape analysis system based on MD simulation. The calculation of free energy is of great importance for understanding the kinetics and the structural determinants of biomolecular processes, such as the folding and unfolding of proteins, ligand bindings to receptors and enzymes, and the transport of small molecules through channels. Since they require extensive sampling of configuration space, there are many variations of molecular dynamics simulation methods such as unfolding simulations, the replica exchange method [4], and the multi-canonical simulation method to construct free energy landscapes [5]. One important underlying technique is optimization of the MD simulation program.

In this paper, we describe a Free Energy Landscape Analysis System, the relevant aspects of Blue Gene/L architecture, and our tuning and evaluation of its performance in an MD simulation. We are distributing our acceleration patch program from our website (www.cbrc.jp/cbrc/intro/cluster/BlueProtein.eng.html).

2. Outline of Free Energy Landscape Analysis System

In this system (**Fig. 1**), first trajectory analysis was carried out to set the free energy of the reaction coordinates after the conformation sampling using MD simulation on the Blue Gene/L. Second, estimation of the probability density was done along a given coordinate from the set of configurations generated via simulation. Then the free energy was calculated from the probability density of each configuration. Finally, all of the calculated free energy data was plotted using gnuplot. These steps were done using a Linux server for the Free Energy Landscape Analysis System. The free energy was determined by calculating probabilities from a histogram analysis. The value of the

†1 Computational Biology Research Center, National Institute of Advanced Industrial Science and Technology (AIST)
†2 Deep Computing Development Laboratory, IBM Japan Ltd.
†3 Institute for Biological Resources and Functions, National Institute of Advanced Industrial Science and Technology (AIST)
†4 Department of Computer Science, Graduate School of Information Science and Engineering, Tokyo Institute of Technology

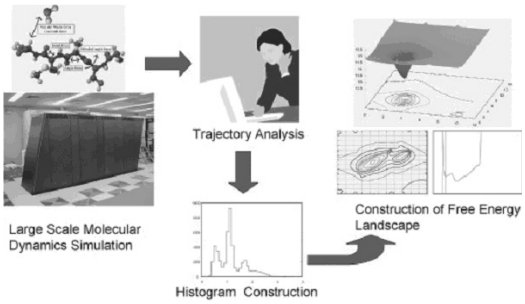


Fig. 1 Outline of Free Energy Landscape Analysis System.

free energy of a conformation is $F = -RT \ln P$, where P is the probability that a conformation exists, R is gas constant, and T is temperature. This system handles multiple trajectories from MD simulations, and is applicable for unfolding simulations, replica exchange simulations, and multi-canonical simulations, and constructs a histogram which shows the conformational distribution. Currently, this system automatically constructs and analyzes the free energy landscape of (i) the Radius of Gyration (R_G), (ii) the End-to-End distance, (iii) the RMSD (Root Mean Square Deviation) from the initial structure, and (iv) the distances among any residues that the user selects based on trajectory analysis. We computed the R_G parameter as

$$R_G = \sqrt{\frac{1}{2N^2} \sum_{i,j}^{N} (\vec{r_i} - \vec{r_j})^2},$$

considering only the coordinates $\vec{r_n}$ of the $C\alpha$-atoms of all of the N residues.

3. Blue Gene/L and Tuning Policy

3.1 Specifications of the Blue Gene/L Architecture

The IBM® System Blue Gene® Solution is a massively parallel supercomputer based on IBM's system-on-chip technology. It is designed to scale to 65,536 dual-processor nodes, with peak performance of 360 teraflops. Each core in the system has a pair of IBM Power PC® (PPC 440) processors with two floating-point units (FPU) produced in 130-nm copper IBM CMOS technology [6]. To achieve a high level of integration and large quantity of microprocessors with low power consumption, the machine was developed based on a processor with a moderate frequency. Each processor core runs at a frequency of 700 MHz giving a theoretical peak performance of 2.8 Gflops/core,

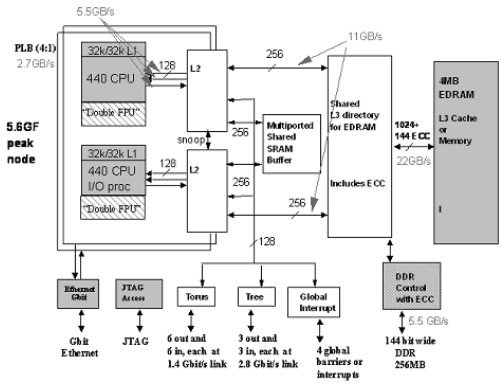


Fig. 2 Compute node ASIC of Blue Gene.

and 5.6 Gflops/chip since each chip includes two cores. Each chip constitutes a compute node. Each node is very simple, consisting of a single ASIC containing two processor cores and double-data-rate (DDR) SDRAM chips of 1-Gbyte capacity (**Fig. 2**). The nodes are interconnected through five networks, the most important of which is a three-dimensional torus network that has the highest aggregate bandwidth and handles the bulk of communications. There are virtually no asymmetries in this interconnect, so the nodes communicate with neighboring nodes that are physically close on the same board and with nodes that are physically far removed on a neighboring rack with the same bandwidth and nearly the same latency. This allows for a simple programming model because there are no edges in a torus configuration. The SoC ASIC that contains the node incorporates all of the functionality needed by Blue Gene. It also contains a 4-MB L3 cache of extremely high-bandwidth embedded DRAM that is on the order of 30 cycles from the registers for most L1/L2 cache misses. The next building block is the compute card. Two compute nodes attached to a processor card with memory (RAM) make up a compute card. The I/O card is the next building block. This card is very similar to the compute card. The I/O card has an integrated Ethernet connection for communicating with the outside world. Compute cards and I/O cards are plugged into a node card. There are two rows of eight compute cards on the node card. A midplane consists of 16 node cards stacked in a rack. One rack holds two midplanes, for a total of 32 node cards consisting of 1,024 compute nodes interconnected by a 3D torus.

Some aspects of the machine which are in-

teresting to application programmers are summarized below. First, there are two identical IBM PowerPC 440 CPU cores on each chip, and each CPU has a dual floating-point unit (FPU). This FPU executes instructions in single-instruction multiple-data (SIMD) fashion, like a two-element vector processor. Making effective use of these double-FPUs is one of the key strategies for application performance improvement. Although the two CPU cores are identical, one of the CPU cores is intended as a communication coprocessor when executing communication-intensive applications. There are two main modes of operation supported by the system software: each processor can handle its own communication (virtual node mode), or one processor can be dedicated to communication and one to computation (communication coprocessor mode). Making effective use of the virtual node mode is another key aspect for the performance improvement. BG/L has two physically separate networks used for high-performance communication between nodes: a 3D torus network in which every node is connected to its six nearest topological neighbors, and a hardware collective network that allows rapid broadcasts and reductions. The collective hardware supports fixed-point operations within reductions. Making effective use of these networks and related hardware functions is also important for performance.

3.2 Tuning Policy of Blue Gene

3.2.1 Enabling the Double-FPU

To maximize the performance we need to use as many double-FPU instructions as possible. The double-FPU can do two floating point calculations at once, so the double-FPU is good at handling even number data sets such as 2-dimensional or 4-dimensional vector calculations or complex arithmetic. In particular, the double-FPU has a special instruction set for complex arithmetic. For example, the multiplication of two complex values can be calculated by two double-FPU instructions [7].

For such even-number data sets or complex arithmetic, the IBM XL compiler can generate double-FPU instructions automatically in many cases. However, for odd number data sets such as 3-dimensional vector calculations, the IBM XL compiler can rarely generate double-FPU instructions automatically. For odd number data sets, if the data and calculations can be handled sequentially, it is a possible to use double-FPU instructions. To allow the compiler to generate double-FPU instructions for sequential data, it is better to use 1-dimensional arrays instead of the original array structure used in the subroutine. The following modifications of the source code allow the compiler to generate double-FPU instructions for a 3-dimensional vector calculation.

- : original source code
- REAL*8 va(3,N),vb(3,N),t
- DO I=1,N
- va(1,I)=va(1,I) + t * vb(1,I)
- va(2,I)=va(2,I) + t * vb(2,I)
- va(3,I)=va(3,I) + t * vb(3,I)
- ENDDO

- : modified source code
- REAL*8 va(3*N),vb(3*N),t
- DO I=1,N*3
- va(I)=va(I) + t * vb(I)
- ENDDO

3.2.2 Enabling Parallel Load and Store Instructions

To feed two floating point values into a double-FPU, there are special load and store instructions which can load two sequential values from an array into a double-FPU register or store two sequential values from a register into an array at the same time. We can improve the performance of programs by using these parallel load and store instructions as much as possible. Parallel load and store decrease the time to load and store, and also these instructions are good for improving instruction scheduling.

To execute parallel load and store instructions, there are two limitations [8]:

- the two values in the array must be sequential
- the two values must be aligned on a 16-byte boundary

If the two values are separated, for example in different arrays, we can not use parallel load and store instructions. In this case, we should use two of the usual load and store instructions. The second limitation is very critical to performance. If the two values are stored at an address crossing over a 16-byte boundary, the parallel load and store instructions can still be used but it takes thousands of clocks to handle the irregular procedure call in software.

The IBM XL compiler always allocates arrays to be aligned on 16-byte boundaries. Thus we need to be concerned about 16-byte boundaries only for even number data sets or for sequential calculations on 1-dimensional arrays.

Unfortunately we have to take care of the 16-byte boundaries if the data structure is odd-numbered or if the data access is non-sequential in a 1-dimensional array.

The IBM XL compiler can detect whether or not the data is aligned and can generate code for each case automatically for simple sequential calculations. However in most cases, the compiler can not detect the 16-byte alignment for a given array in subroutines. To make the compiler aware of the 16-byte alignment, the *alignx* intrinsic function is used for arrays that are known (by the programmer) to be guaranteed to be 16-byte boundary aligned. Given appropriate use of *alignx*, the compiler can generate parallel load and store instructions for computations which would otherwise escape automatic detection.

3.3 Profiling and Tuning Amber 9 on Blue Gene

3.3.1 Porting Amber 9 on Blue Gene

(i) Compiling SANDER program

Although we adopted a parallel SANDER (Simulated Annealing with NMR-Derived Energy Restraints) module that is one of the MD programs in Amber 9 [9], the elements mentioned below will apply to other MD programs. This simulation of a continuous process is broken down into small discrete time steps, each of which is an iteration of two parts: a force calculation (calculating the forces from the evaluated conformational energies) and an atom update (calculating the new coordinates of the molecules).

SANDER already uses the Message Passing Interface (MPI) [10] for data communication, so it is easy to port this program to Blue Gene. There is an automatic configuration script in the Amber 9 package, but unfortunately there is no setting for Blue Gene, so to create a configuration for Blue Gene, we ran the configuration script with "./configure -mpich xlf90_suse" and we modified the config.h file generated by the configuration script.

We used the IBM XL FORTRAN compiler v.10.1 for Blue Gene to build SANDER. The compiler optimization options we used were "-O3 -qhot -qstrict -qarch=440d -qtune=440".

(ii) Increasing the limits of parallelization

SANDER limits the maximum number of processors to 256 processors in the default settings. We can increase the limit by modifying the value of "MPI_MAX_PROCESSORS" defined in "parallel.h" and "ew_parallel.h" for an appropriate number of processors. Then we must add the data for the array "logtwo" defined in "parallel.h".

3.3.2 Profiling SANDER on Blue Gene

Before we tune a program for Blue Gene, it is important to know the characteristics of the program to plan the tuning strategy. SANDER includes its own timing routines and can display timing information for each module in the program. We used this information for profiling SANDER on Blue Gene.

We ran SANDER in coprocessor mode. In coprocessor mode, we can only use one processor core in a compute node for calculations while the other processor core is used for the MPI functions. There is another mode called virtual node mode, in this mode, we can use both processor cores for calculation, and these processor cores can be treated as independent MPI tasks. Thus virtual node mode potentially has double the computational power, but the actual performance is not always twice that of coprocessor mode, because in virtual node mode the L3 cache and memory bandwidth are shared by two processor cores, and each processor core has to handle MPI functions. The following table shows comparisons of the elapsed time for the prion data set (see Results section) between coprocessor (CO) mode and virtual node (VN) mode.

Comparing the times with the same numbers of nodes, VN mode is faster, but comparing times with the same number of processors, CO mode is faster. Therefore, if we can only use a limited number of nodes, then it is better to choose VN mode, but if we can use as many nodes as the program's scalability will allow, then CO mode will be better (**Table 1**), so we choose CO mode to run SSANDER.

We tested various numbers of processors for profiling. We used a prion data set that contains 31,562 atoms. Most of the calculations performed by SANDER are the calculations of the forces between atoms. Using molecu-

Table 1 Comparison of the elapsed time between coprocessor mode and virtual node mode.

	CO mode	VN mode
16 nodes (32 procs. in VN mode)	181.02 sec.	136.61 sec.
32 nodes (64 procs. in VN mode)	116.08 sec.	98.13 sec.
64 nodes (128 procs. in VN mode)	92.43 sec.	

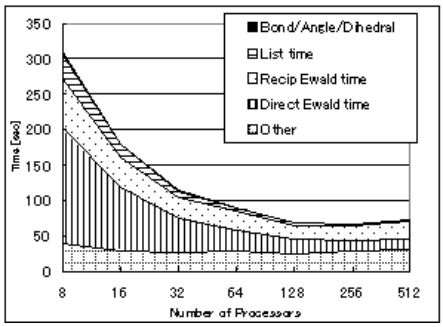


Fig. 3 Profiling result of prion data set. This graph shows the timing of major 4 modules and other procedures (lumped together) of SANDER.

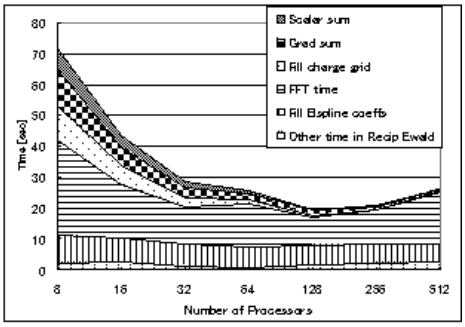


Fig. 4 Profiling result of reciprocal Ewald force calculation.

lar topology, it is easy to calculate forces between two connected atoms. These forces are called the bonding forces, shown in **Fig. 3** as "Bond/Angle/Dihedral". These calculations are not too heavy and the scalability is good. The forces between non-connected atoms are called non-bonding forces. When calculating the non-bonding forces, there is no topology information, so the program has to collect a list of all of the atoms that may affect each other at the beginning of each time step. This module is shown as "List time". This module is also well parallelized. Non-bonding forces are calculated by the two modules shown as "Recip Ewald time" and "Direct Ewald time". "Direct Ewald time" shows good scalability but "Recip Ewald time" is not so scalable as the number of processors increases. The "Other" category shows no scalability, because there are non-parallelized procedures in "Other". These non-parallelized procedures limit the scalability of the program.

Figure 4 shows detailed profiling results for the "Recip Ewald time".

"Scalar sum", "Grad sum", and "Fill Charge grid" are very well parallelized but the other procedures are not. The heaviest module in the reciprocal Ewald force calculation is the FFT module and the scalability of FFT is saturated by 128 processors. This FFT module is included in SANDER but is not tuned well for any architecture.

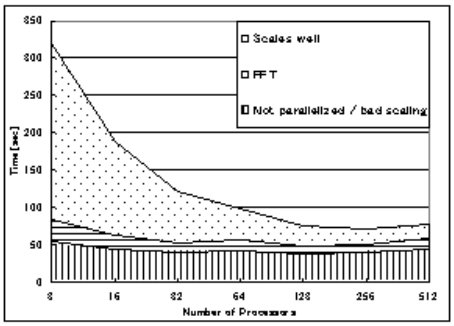


Fig. 5 The execution times of procedures: sum of scaled procedures, FFT routine, and sum of non parallelized or non scalable procedures are shown.

3.3.3 Tuning Study of SANDER on Blue Gene

Figure 5 shows another profiling result that compares the time of some procedures: scaled, FFT, and non-scaled. In this graph, the non-scaled procedures consist of non-parallelized procedures and procedures for which the scalability is bad.

Figure 5 shows the limitations of the performance improvement of SANDER. If we tune the scaled procedures, the performance will be greatly improved when we use a small number of processors; but the performance improvement in terms of the total time will be small when we use a large number of processors. If we can tune the non-scaled procedures, the scalability of SANDER will be improved for a larger number of processors. FFT is one of the heaviest non-scaled procedures and potentially there is some room to tune it for the Blue Gene architecture. Most of the other non-scaled procedures can not be parallelized. If we can tune the local performance of these procedures, the scalability will not be changed, but the overall time will be decreased.

For both cases, scaled and non-scaled procedures, we tuned the procedures by exploiting the double-FPU as much as possible. SANDER is well parallelized except for the FFT module, and it is difficult to modify non-parallelized procedures so that they are parallelized well on Blue Gene. Thus our tuning of SANDER fo-

cused on two points: (i) enabling double-FPU operations and (ii) improving the FFT module.

(i) Enabling double-FPU operations

Most of the calculations of the MD programs are 3-dimensional vector calculations, because the properties of atoms such as their positions, velocities, and forces are presented as 3-dimensional vectors. As we described in Subsection 3.2.1, the double-FPU is not good at handling odd-numbered data structures such as 3-dimensional vector calculations. To generate double-FPU instructions for MD calculations, we merge two 3-dimensional vectors and let the two vectors be calculated together. However in most case in SANDER, it is very difficult to merge pairs of calculations. We tried to enable double-FPU operations in many cases, but most of the modifications were not effective in improving performance because these calculations were not critical to the overall performance according to the profiling results. Therefore we selected some functions where merging pairs of calculations was possible and effective in boosting the performance.

SANDER has its own array allocation routine that allocates the array by sharing a part of the large array stack allocated when the program starts. When the IBM XL compiler allocates an array, the address of the top of the array is always aligned to a 16-byte boundary. However the array allocation routine of SANDER was not designed to check for 16-byte boundaries, so we modified this routine to returns allocated arrays whose address is always aligned to a 16-byte boundary. With this modification and by inserting the ALIGNX intrinsic function, double-FPU instructions can be generated for simple cases involving sequential access.

(ii) Improving the FFT module

The FFT module of SANDER is a complex 3-dimensional FFT, so an external FFT library could be used instead of the original FFT module, and this seemed to be the best way to improve the performance of the FFT. But the original FFT module is tightly dependent on the rest of SANDER for memory allocation and data structures. Therefore, for the current research, we did not use an external FFT library, but instead we tuned the original FFT module to support double-FPU operations.

In an FFT module, most of calculation is complex arithmetic, which is suitable for the double-FPU. Unfortunately, the original source code does not use the complex data type, but uses the real data type so the compiler cannot generate double-FPU instructions. We modified the code to use complex data types and the double-FPU intrinsic functions to generate appropriate instructions.

The original FFT module used MPI_ISEND and MPI_RECV to exchange data between processors for the array transpose. We modified the transpose procedure and we used MPI_ALLTOALL to exchange all of the data at once, because we ran the program using the coprocessor mode of Blue Gene. In the coprocessor mode the performance of MPI_ALLTOALL is better than using MPI_ISEND and MPI_RECV for all of the pairs of processors.

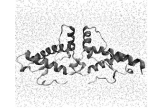
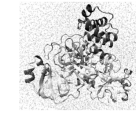
4. Results

4.1 Performance Evaluation of MD simulation

We measured the simulation time of SANDER for two different protein data sets (**Table 2**): the prion protein and the SARS coronavirus main proteinase. These structures were obtained from PDB. The prion protein is associated with the infection of Prion diseases such as BSE, the new variant CJD, and scrapie. The SARS coronavirus main proteinase is a key structure of the SARS coronavirus and plays an important role in the virus lifecycle through the specific processing of viral polyproteins. There are disulfide bonds between Cys179-Cys311 and Cys214-Cys287 in the dimeric prion protein. The systems were surrounded with a layer of TIP3P water molecules using the LeaP program. The number of solvent water molecules and counter ions in each system are shown in Table 2.

For each data set we set the iteration count to 1,000. **Figure 6** shows the elapsed time measurements for these two data sets. In this

Table 2 Description of data sets for Performance Evaluation.

Prion protein (PDBID:1I4M [11])	SARS coronavirus main proteinase (PDBID:1UJ1 [12])
216 residues (3,440 atoms) 9,374 water molecules total: 31,562 atoms	605 residues (4,675 atoms) 24,212 water molecules total: 77,314 atoms
	

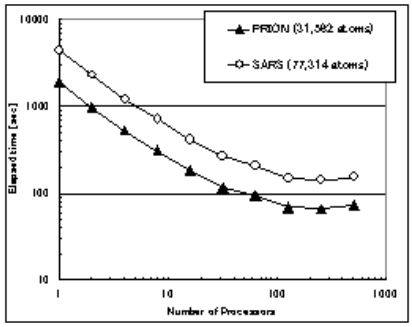


Fig. 6 Time measurement results of simulating two protein data sets with the original SANDER program on Blue Gene.

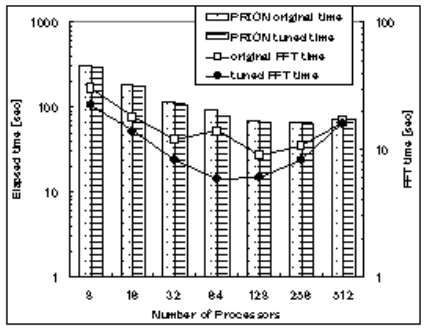


Fig. 7 Comparisons of measured simulation times of the prion data set. Bar graphs denote total simulation times, and line graphs denote FFT times.

measurement we used the non-tuned original SANDER.

Both data sets showed very similar scalability in Fig. 6, reaching saturation near 128 processors. The performance gain slowly decreased over 64 processors. Therefore we selected 32 processors as a practical number of processor for MD simulation with SANDER for data sets with similar numbers of atoms, because we can run more test cases on different partitions with 32 processors than when running one test case on a large partition. We originally expected that the SARS data would scale up to more processors. However our measurements show almost the same scalability as the prion data set. We think this scalability limitation is due to the implementation of SANDER, and **Fig. 7** effectively illustrates this problem. Non-scaled or non-parallelized modules are very dominant if the number of processors is large. The percentage of scaled modules can be increased if the number of atoms is large enough, but for this we need at least 10 times as large a data set as the SARS data set. Otherwise, we should use

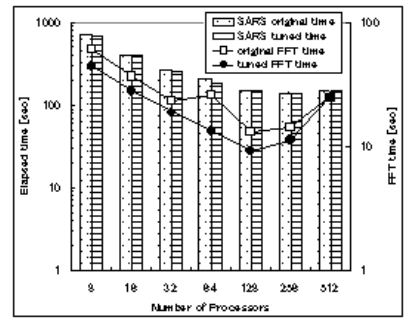


Fig. 8 Comparisons of measured simulation times of the SARS data set.

another MD implementation for problem sizes similar to those measured in this paper.

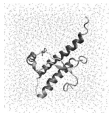
As we noted in Subsection 3.3.3, we modified SANDER to enable double-FPU operations, and in particular we tuned the FFT module. We compared the simulation times between the original SANDER and the tuned SANDER. Figure 7 and **Fig. 8** show the comparison results of the prion data set and the SARS data set, respectively.

We increased the performance of the FFT module by about 30–40%, and we also increased the scalability limit of the FFT module to up to 64 processors for the prion data set, and up to 128 processors for the SARS data set. Even though we tuned the FFT module, the overall performance was little improved. We also modified other modules to enable double-FPU operations, but we could not enable them for all of the calculations and the performance did not improve. For significant improvement it appears to be necessary to optimize the overall algorithm or data structures for the double-FPU architecture.

4.2 Free Energy Landscape Construction

We constructed free energy landscapes for two different protein data sets (See **Table 3**): the prion protein and the chignolin protein. The chignolin protein was designed, synthesized, and had its structure determined by one of the authors [14]. We used 64 processors to calculate both data sets by using coprocessor mode. The prion data was evaluated 20 times for 30 nanoseconds with different initial velocities in simulations at 300°K. The chignolin data was evaluated for 1-microsecond simulations at 315°K. Although Table 3 shows the folded structure of the chignolin protein (PDBID:1UAO), we executed the simu-

Table 3 Description of the data sets for Free Energy Landscape Construction.

Prion protein (PDBID:1XYX [13])	Chignolin protein (PDBID:1UAO)
112 residues (1,794 atoms) 3,868 water molecules 1 sodium ion total: 13,399 atoms	10 residues (138 atoms) 2,270 water molecules 2 sodium ions total: 6,950 atoms
	

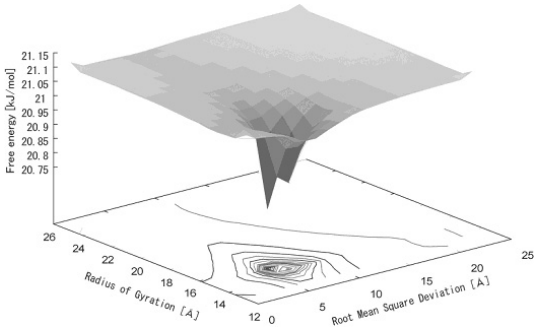


Fig. 9 Free energy landscape of mouse Prion protein along R_G and RMSD.

lation from the elongated state in the simulations. There are disulfide bond between Cys170-Cys222 in the prion protein. The systems were surrounded with layer of TIP3P water molecules using the LeaP program. The numbers of solvent water molecules and counter ions in each system are shown in Table 3.

Figure 9 shows plots of the free energy landscape of the mouse Prion protein. The value of R_G gives an estimate of the characteristic volume of a globular polymer, which is inversely related to compactness. Figure 9 suggests the mouse Prion protein has two stable states (conformations). This simulation required approximately 19,200 days of CPU time.

Figure 10 shows plots of the free energy landscape of the chignolin protein. In the figure, the distances of Asp3:O-Gly7:N and of Asp3:O-Thr:8:N were used as axes, because the hydrogen bonds between these atoms are believed to be related to the essential interactions of the chignolin protein. Honda, et al. have experimentally determined that the chignolin protein does not have a large energy gap between its folded and unfolded states as compared with a typical wild protein. Although this was confirmed in past research [15),16)], our results also

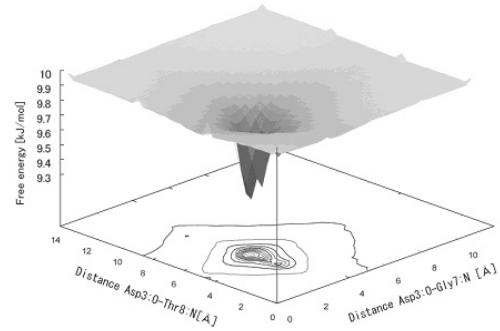


Fig. 10 Free energy landscape of chignolin protein along the distance between Asp3:O-Gly7:N vs. Asp3:O-Thr:8:N.

show the effectiveness of our method in obtaining the free energy landscape even for this kind of small artificial protein. This simulation required approximately 9,600 days of CPU time. The details of these free energy landscape analyses will be published for the protein science literature. These kinds of analyses will support rapid progress in life science based on the computer science contribution.

5. Conclusion

In this paper, we discussed a Free Energy Landscape Analysis System and optimization of an MD program on Blue Gene/L. We improved the total scalability up to 64 processors. Now we can effectively use 64 processors for MD simulation for both cases. However although the scalability of the FFT module for the SARS data set was greatly improved, the total time was not improved compared to 128 processors. There remains room to tune modules other than the FFT module that currently do not scale well with more processors. We modified other modules to support double-FPU operations, but actual double-FPU instructions were not generated by the compiler in many cases. Currently, there are many 3-dimensional vector calculations for which it is difficult to induce the compiler to generate double-FPU instructions. Even in the FFT module, there are some cases for which we could not figure out why the compiler was unable to generate parallel load and store instructions for arrays that were clearly aligned to 16-byte boundaries. We intend to further analyze the MD program and the code generated by the compiler and feed this analysis back into the further development of the Blue Gene compiler. We also showed the free energy landscapes of two data sets, the

prion protein and the chignolin protein. We suggested that the prion protein has two stable states in conformational space from the free energy landscape of R_G vs. RMSD, and our method is also applicable to small artificial proteins such as the chignolin protein which has a small energy gap between the native and denatured states.

It is still necessary to improve the simulation performance to obtain biological results within realistic amounts of time. We plan to apply Space Decomposition (SD) [17] in which the simulation domain is usually broken into P (number of processors) subdomains and each processor computes forces only on the atoms in its subdomain or apply other paradigms [18] for similar optimizations.

In addition, we are planning to consider additional analysis methods which automatically detect salt-bridges and similarities between free energy landscapes.

Acknowledgments We thank Dr. K. Ikeda at PharmaDesign Inc. for his helpful advice on free energy landscape analysis.

References

1) Berman, H.M., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T.N., Weissig, H., Shindyalov, I.N. and Bourne, P.E.: The Protein Data Bank, *Nucleic Acids Research*, Vol.28, pp.235–242 (2000).
2) Karplus, M. and Kuriyan, J.: Molecular Dynamics and Protein Function, *Proc. Natl. Acad. Sci. USA.*, Vol.102, pp.6679–6685 (2005).
3) Sekijima, M., Motono, C., Yamasaki, S., Kaneko, K. and Akiyama, Y.: Molecular dynamics simulation of dimeric and monomeric forms of human Prion protein: Insight into dynamics and properties, *Biophysical Journal*, Vol.85, pp.1176–1185 (2003).
4) Sugita, Y. and Okamoto, Y.: Replica-exchange molecular dynamics method for protein folding, *Chemical Physics Letters*, Vol.314, pp.141–151 (1999).
5) Nakajima, N., Nakamura, H. and Kidera, A.: Multicanonical ensemble generated by molecular dynamics simulation for enhanced conformational sampling of peptides, *J. Phys. Chem.,* Vol.B101, pp.817–824 (1997).
6) Gara, A., et al.: Overview of the Blue Gene/L system architecture, *IBM Journal of Research and Development*, Vol.49, No.2/3, pp.195–210 (2005).
7) Doi, J., Samukawa, H., Matsufuru, H. and Hashimoto, S.: High Parallelization of Lattice QCD program for Blue Gene, *IPSJ High Performance Computing Symposium 2006*, Tokyo, Japan, pp.87–94 (2006).
8) Chatterjee, S., et al.: Design and exploitation of a high-performance SIMD floating-point unit for Blue Gene/L, *IBM Journal of Research and Development*, Vol.49, No.2/3, pp.377–391 (2005).
9) Case, D.A., Darden, T.A., Cheatham, III, T.E., Simmerling, C.L., Wang, J., Duke, R.E., Luo, R., Merz, K.M., Pearlman, D.A., Crowley, M., Walker, R.C., Wang, B., Hayik, S., Roitberg, A., Seabra, G., Wu, X., Brozell, S., Tsui, V., Gohlke, H., Yang, L., Tan, C., Mongan, J., Hornak, V., Cui, G., Beroza, P., Mathews, D.H., Schafmeister, C., Ross, W.S. and Kollman, P.A.: AMBER 9, University of California, San Francisco (2006).
10) Message Passing Interface Forum, MPI: A Message Passing Interface Standard. http://www-unix.mcs.anl.gov/mpi/
11) Knaus, K.J., Morillas, M., Swietnicki, W., Malone, M., Surewicz, W.K. and Yee, V.C.: Crystal structure of the human prion protein reveals a mechanism for oligomerization, *Nat. Struct. Biol.*, Vol.8, pp.770–774 (2001).
12) Yang, H., Yang, M., Ding, Y., Liu, Y., Lou, Z., Zhou, Z., Sun, L., Mo, L., Ye, S., Pang, H., Gao, G.F., Anand, K., Bartlam, M., Hilgenfeld, R. and Rao, Z.: The crystal structures of severe acute respiratory syndrome virus main protease and its complex with an inhibitor, *Proc. Natl. Acad. Sci. USA.*, Vol.100, pp.13190–13195 (2003).
13) Gossert, A.D., Bonjour, S., Lysek, D.A., Fiorito, F. and Wuthrich, K.: Prion protein NMR structures of elk and of mouse/elk hybrids, *Proc. Natl. Acad. Sci. USA.*, Vol.102, pp.646–650 (2005).
14) Honda, S., Yamasaki, K., Sawada, Y. and Morii, H.: 10-residue folded peptide designed by segment statistics, *Structure*, Vol.12, pp.1507–1518 (2004).
15) Seibert, M.M., Patriksson, A., Hess, B. and van der Spoel, D.: Reproducible polypeptide folding and structure prediction using molecular dynamics simulations, *J. Mol. Biol.*, Vol.354, pp.173–183 (2005).
16) Satoh, D., Shimizu, K., Nakamura, S. and Terada, T.: Folding free-energy landscape of a 10-residue mini-protein, chignolin, *FEBS Lett.*, Vol.580, pp.3422–3426 (2006).
17) Fincham, D.: Parallel computers and molecular simulation, *Molecular Simulation*, Vol.1, pp.1–45 (1987).
18) Sekijima, M., Takasaki, S., Nakamura, S. and Shimizu, K.: Automatic Improvement of

Scheduling Policies in Parsley Parallel Programming Environment, *Proc. 14th IASTED International Conference on Parallel and Distributed Computing and Systems*, pp.380–385 (2002).

(Received February 19, 2007)
(Accepted March 30, 2007)

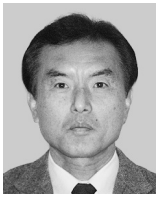
(Communicated by *Tatsuya Akutsu*)

Masakazu Sekijima received Ph.D. from the University of Tokyo in 2002. Since 2002 he has worked at the National Institute of Advanced Industrial Science and Technology (AIST) as a Research Staff and since 2003 as a Research Scientist. His current research interests are High Performance Computing, Bioinformatics and Protein Science. He is a member of IPSJ, IEEE and ACM.

Jun Doi is a research staff member of IBM Research, Tokyo Research Laboratory since 1999. His current research interests are supercomputing, scientific application tuning especially for Blue Gene.

Shinya Honda works in the National Institute of Advanced Industrial Science and Technology (AIST) as a Leader of Molecular and Cellular Breeding Research Group. He began his career as a governmental research scientist in the Research Institute for Polymer and Textiles in 1988. Then, he moved to the National Institute of Bioscience and Human-Technology in 1993. Since 2001 he has been in AIST. In parallel with his governmental research, he received his Ph.D. degree in applied chemistry from Waseda University in 2000. His current research interests are in analytical biophysics, protein design, and synthetic biology.

Tamotsu Noguchi received Ph.D. from Osaka University in 2001. He had been in Fujitsu Limited, Real World Computing Partnership as a senior research scientist, and PharmaDesign, Inc. as a senior research scientist. Since 2001 he has been in National Institute of Advanced Industrial Science and Technology (AIST). Since 2001 he has been in AIST as a senior research scientist, since 2003 as a team leader, since 2007 as a deputy director. His current research interests are disorder regions in a protein and prediction of protein functions.



Shigenori Shimizu is a senior technical staff member of IBM, working in computer architecture area. He received B.E., M.S., and Ph.D. degrees, all in instrumentation engineering, from Keio University in 1977, 1979, and 1983, respectively. His areas of specialization include parallel processing systems, VLSI architecture and design, and HPC applications. He is a member of IPSJ and IEICE.

Yutaka Akiyama received Dr. Eng. from Keio University in 1990. He served as the director of Computational Biology Research Center (CBRC), National Institute of Advanced Industrial Science and Technology (AIST), from 2001 to 2007. He became a full professor at Department of Computer Science, Tokyo Institute of Technology in April 2007. His research interest covers parallel processing and computational biology, including protein structure analysis, protein docking, mass spectrometry and pharmacokinetics. He is a member of IPSJ.