Available online at www.sciencedirect.com
ScienceDirect
J. Parallel Distrib. Comput. 68 (2008) 678–685
Journal of Parallel and Distributed Computing
www.elsevier.com/locate/jpdc

Parallel solution of large-scale eigenvalue problem for master equation in protein folding dynamics

Yiming Li[a,*], Shao-Ming Yu[b], Yih-Lang Li[b]

[a]*Department of Communication Engineering, National Chiao Tung University, Hsinchu, Taiwan*
[b]*Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan*

Received 5 May 2007; received in revised form 24 August 2007; accepted 12 September 2007
Available online 22 September 2007

Abstract

It is known that a master equation characterizes time evolution of trajectories and transition of states in protein folding dynamics. Solution of the master equation may require calculating eigenvalues for the corresponding eigenvalue problem. In this paper, we numerically study the folding rate for a dynamic problem of protein folding by solving a large-scale eigenvalue problem. Three methods, the implicitly restarted Arnoldi, Jacobi–Davidson, and QR methods are employed in solving the corresponding large-scale eigenvalue problem for the transition matrix of master equation. Comparison shows that the QR method demands tremendous computing resource when the length of sequence $L > 10$ due to extremely large size of matrix and CPU time limitation. The Jacobi–Davidson method may encounter convergence issue, for cases of $L > 9$. The implicitly restarted Arnoldi method is suitable for solving problems among them. Parallelization of the implicitly restarted Arnoldi method is successfully implemented on a PC-based Linux cluster. The parallelization scheme mainly partitions the operation of matrix. For the Arnoldi factorization, we replicate the upper Hessenberg matrix $\mathbf{H_m}$ for each processor, and distribute the set of Arnoldi vectors $\mathbf{V_m}$ among processors. Each processor performs its own operation. The algorithm is implemented on a PC-based Linux cluster with message passing interface (MPI) libraries. Numerical experiment performing on our 32-nodes PC-based Linux cluster shows that the maximum difference among processors is within 10%. A 23-times speedup and 72% parallel efficiency are achieved when the matrix size is greater than 2×10^6 on the 32-nodes PC-based Linux cluster. This parallel approach enables us to explore large-scale dynamics of protein folding.

Keywords: Master equation; Eigenvalue problem; Implicitly restarted Arnoldi method; Jacobi–Davidson method; QR method; Parallelization; PC-based Linux cluster; MPI

1. Introduction

Dynamics of protein folding has recently been of great interest. It is nowadays explored in different way, such as mass action models, all atoms, lattice, and off-lattice model, and methods between macroscopic and microscopic models [1,2,3,4,5,6,7,11,15,20,21,22]. One of biopolymer folding dynamic core problems is finding an ensemble of transition state conformations or rate-limiting steps. For small molecules, numerical methods theoretically could find transition states, i.e. the saddle points of the potential energy surface. However, for proteins or RNA macromolecules, the potential energy surface of them is usually more complicated and has many pathways. Among approaches, a master equation [8,5] is crucial in dynamic simulation of protein folding. Deriving from the Liouvillian, the master equation basically describes time evolution of a distribution of trajectories. Numerical solution of the master equation requires computing eigenvalues λ_N and eigenvectors for the corresponding N by N transition matrix. Those negative and larger eigenvalues determine the slowest dynamical relaxation processes. The size of transition matrix grows dramatically with respect to the length of studied protein, which results in a large-scale eigenvalue problem to be solved.

In this study, we first compare the QR [28,17], implicitly restarted Arnoldi [24,27], and Jacobi–Davidson [12,23] methods in calculating all or first few larger nonpositive eigenvalues of the matrix arising from the master equation in protein folding problems. These methods are implemented and tested with respect to different problem size. For a problem with more

* Corresponding author. Fax: +886 3 572 6639.
E-mail address: ymli@faculty.nctu.edu.tw (Y. Li).

0743-7315/$ - see front matter © 2007 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2007.09.002

than 10 residues, the QR method is the most stable algorithm in solving whole eigenvalues of the matrix, but it seriously encounters huge computing resource troubles. It requires large-size of memory to store the whole matrix and the CPU time grows exponentially. It becomes impossible for us to apply the QR method to solve a realistic protein folding problem which involves more than 10 residues. For a problem with 9 residues, the Jacobi–Davidson method can only solve 26 eigenvalues if we want to compute the first 50 eigenvalues, for example. By applying a sparse matrix technique to the transition matrix, the implicitly restarted Arnoldi method does work for all testing cases. For various testing proteins with 16 and 17 residues, the corresponding eigenvalue problems with N in the order of millions are solved successfully, where the first few computed nonpositive eigenvalues are discussed. Our comparison suggest that the implicitly restarted Arnoldi method is robust in calculating the desired eigenpairs of the master equation. Unfortunately, the calculation of eigenvalue is a time-consuming task and requires a lot of computational resources. Therefore, parallelization of the implicitly restarted Arnoldi method will benefit the investigation of protein folding dynamics with large sequences.

We further parallelize the implicitly restarted Arnoldi method and for the first time introduce this method to explore dynamics of protein folding. Our parallelization scheme principally partitions the operations of the matrix. For the Arnoldi factorization, we replicate the upper Hessenberg matrix $\mathbf{H_m}$ for each processor, and distribute the set of Arnoldi vectors $\mathbf{V_m}$ among processors. Each processor performs its own operations. The parallel technique distributes the request of computing resources among PCs uniformly and accelerates the calculation of eigenvalues λ_N significantly. In terms of several computational benchmarks [18,19,10,9], we test the parallel algorithm with respect to different size of problem on our constructed 32-nodes PC-based Linux cluster with message passing interface (MPI) libraries. For proteins with 16 and 17 residues, parallel solution of the eigenvalue problem with N in the order of millions is successfully implemented, where the accuracy of computed nonpositive eigenvalues is almost the same with the solution computed from a single PC. Results of parallel performance are achieved in terms of the load balancing, speedup and efficiency. The scale of parallel performance is examined for the aforementioned benchmarks with respect to different matrix size. For small number of processors, the efficiency is almost independent of the matrix size. For the number of processors is greater than 8, the efficiency increases when the matrix size increases; in particular, the scale is evident and continuously improved even for the matrix size greater than 10^5 on the 32-nodes PC-based Linux cluster. A 23-times speedup and over 72% parallel efficiency are simultaneously observed when the matrix size is greater than 2×10^6 on the 32-nodes PC-based Linux cluster. According to our numerical examinations, this parallel implementation successfully solves the problem arising from the master equation of protein folding in a good manner. Therefore, it significantly reduces the computational time and effectively governs large-scale eigenvalue problems in bioinformatics.

This paper is organized as follows. In Section 2, we state the master equation and expound the parallel algorithm. In Section 3, we show the results and discussion. Finally, we draw the conclusions.

2. The computational model and parallelization

The master equation describing time evolution of trajectory (i.e., the transition of states) is expressed as [7,8]

$$\frac{dP(t)}{dt} = AP(t). \tag{1}$$

To study the dynamics of Eq. (1), we have to compute the eigenvalues (in general, the first few largest nonpositive eigenvalue) of the matrix A, where $P(t)$ is the N-dimensional vector of the instantaneous probability of the N conformations. Depending on different free energy, A is a sparse and asymmetric N by N transition (or rate) matrix, where the matrix entries is defined as

$$A_{ij} = \begin{cases} k_{i\to j} & \text{for } i \neq j, \\ \sum_{l\neq i} k_{i\to l} & \text{for } i = j, \end{cases} \tag{2}$$

where $k_{i\to j}$ is the rate constant for a protein changes its conformation from the ith conformation to the jth conformation, and only depends on the energy states between two conformations

$$k_{i\to j} = \begin{cases} 1 & \text{for } E_i \geqslant E_j, \\ e^{(E_i - E_j)/kT} & \text{for } E_i < E_j, \end{cases} \tag{3}$$

where E_i and E_j are energy states. k is the Boltzmann constant and T means the temperature. For any initial conformations, solutions of Eq. (1) provides the dynamics of population

$$P(t) = \sum_{m=0}^{N=1} C_m^{(c)} n_m e^{-\lambda_m t}, \tag{4}$$

where the $-\lambda_m$ and n_m are the mth eigenvalue and eigenvector. The overall folding kinetics is the linear combination by a coefficient $C_m^{(c)}$ of N possible protein folding conformations. Therefore, $C_m^{(c)}$ means contribution to overall kinetics from the mth mode. The eigenvalues can be ordered as $\lambda_0 < \lambda_1 \leqslant \lambda_2 \leqslant \cdots \leqslant \lambda_{N-1}$. The eigenvector n_0 for the equilibrium mode gives the equilibrium population distribution. The eigenvector n_1 for the slowest mode indicates the intermediate state of protein folding process which takes longest time to fold to the next conformation.

Furthermore, people may interest in the largest nonpositive eigenvalue λ_1. If there is a large gap between the eigenvalues of slow modes group and a fast modes group, then the overall folding speed is limited by these slow modes. Therefore, this protein may fold fast in the beginning because of the group of fast modes; then there is a rate-determining process because of the group of slow modes, and finally it fold as the ground state conformation with its function. Especially for the eigenvalue spectrum with extremely slow modes and a large gap from other modes, this mode is the bottleneck process of the whole folding process. It suggests that some eigenvectors with

Table 1
The number of conformations of proteins with the length of L residues in a 2D square lattice model

Protein length (L)	Number of conformations (N)
4	5
5	13
6	36
7	98
8	272
9	740
10	2034
15	296,806
16	802,075
17	2,155,667
20	41,889,578

$\lambda_m \ll 0$ disappear quickly. On the other hand, any other eigenvectors with $\lambda_m \sim 0$ will determine the slowest dynamic relaxation processes. The overall folding processing time of the protein is approximated as $1/\lambda_1$. We use HP (hydrophobic and polar residues) model in a 2D square lattice [1,15,8,13,14]. The number of conformations of proteins with L residues is listed in Table 1.

To calculate the first few larger nonpositive eigenvalues (or all eigenvalues) of the constructed matrix above, the QR [28,17], implicitly restarted Arnoldi [24,27], and Jacobi–Davidson [12,23] methods are employed and implemented in our investigation. Among three solution methods, according to our numerical experiment, the implicitly restarted Arnoldi method is suitable for solving the first few larger nonpositive eigenvalues of the corresponding eigenvalue problem above. The implicitly restarted Arnoldi method is known as a direct algorithm for reducing a general matrix into upper Hessenberg form, and it could be more effective than subspace iteration methods for computing the dominant eigenvalues of a large, sparse, real, and asymmetric matrix [17,23,16,6,25]. An algorithm for implementing the implicitly restarted Arnoldi method is shown below.

Arnoldi Algorithm: Make an Arnoldi factorization $\mathbf{AV_m} = \mathbf{V_m H_m} + f_m e_m^T$ While (converge)

(i) Compute the eigenvalues λ_j, $j = 1, 2, \ldots, m$;
(ii) Sorting eigenvalues into a wanted set λ_j, $j = 1, 2, \ldots, k$ and unwanted set λ_j, $j = k+1, k+2, \ldots, m$;
(iii) Perform $m - k = p$ steps of the QR iteration with the unwanted eigenvalues shift to obtain $\mathbf{H_m} Q_m = Q_m \mathbf{H_m^+}$;
(iv) $\mathbf{AV_m} Q_k = \mathbf{V_m} Q_k \mathbf{H_k^+} + f_k^+ e_k^T$, where $\mathbf{H_k^+}$ is the leading principle submatrix of order k for $\mathbf{H_m^+}$;
(v) Extend length k Arnoldi factorization to a length m factorization.

Parallelization scheme used in this work is to partition the operations of the matrix [23,16,6,25]. For the Arnoldi factorization

$$\mathbf{AV_m} = \mathbf{V_m H_m} + f_m e_m^T, \tag{5}$$

where $\mathbf{V_m}$ is the set of Arnoldi vectors, $\mathbf{H_m}$ is the upper Hessenberg matrix, and f_m is the residual vector, it replicates $\mathbf{H_m}$ on every processor. $\mathbf{V_m}$ and f_m are partitioned by rows and are distributed to every processor in the cluster. Explicitly computational steps of the process responsible for the j block are given by

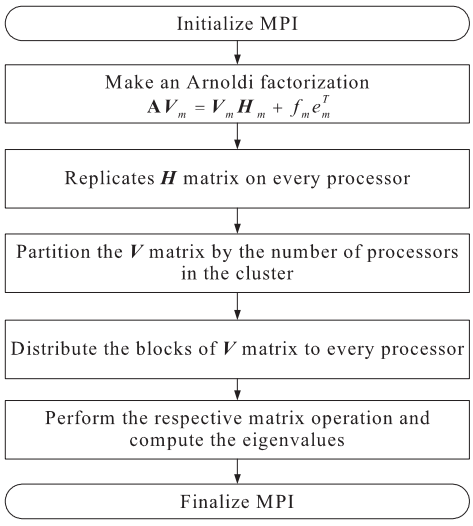

Fig. 1. A computational flowchart of the implemented parallel method.

Step 1. $\beta_m \leftarrow gnorm(||f_m^{(*)}||)$; $v_{m+1}^j \leftarrow f_m^{(j)} \cdot \frac{1}{\beta_m}$;

Step 2. $w^{(j)} \leftarrow A v_{m+1}^j$;

Step 3. $\begin{pmatrix} h \\ \alpha \end{pmatrix}^{(j)} \leftarrow \begin{pmatrix} V_m^{(j)T} \\ V_{m+1}^{(j)T} \end{pmatrix} w^{(j)}$;

$$\begin{pmatrix} h \\ \alpha \end{pmatrix} \leftarrow gsum\left[\begin{pmatrix} h \\ \alpha \end{pmatrix}^{(*)}\right];$$

Step 4. $f_{m+1}^{(j)} \leftarrow w^{(j)} - (V_m, v_{m+1})^{(j)} \begin{pmatrix} h \\ \alpha \end{pmatrix}$;

Step 5. $H_{m+1} \leftarrow \begin{pmatrix} H_m & h \\ \beta_m & e_m^T \end{pmatrix}$; and

Step 6. $v_{m+1}^{(j)} \leftarrow (V_m, v_{m+1})^{(j)}$.

For the steps stated above, β_m is the norm of the distributed vector f_m, v_{m+1}^j is computed by f_m/β_m, and $w^{(j)}$ is the local segment of the matrix–vector product $\mathbf{A}v$ that is consistent with the partition of $\mathbf{V}$. The function *gnorm* at Step 1 is meant to represent the global reduction operation of computing the norm of the distributed vector f_m from the norms of the local segments $f_m^{(j)}$, and the function *gsum* at Step 3 is meant to represent the global sum of the local vectors $h^{(j)}$ so that the quantity $h = \Sigma_{j=1}^{nproc} h^{(j)}$ is available to each process on completion.

Fig. 1 is a computational flowchart for the proposed parallel procedure of the solution method. The communication between processors is based on the MPI, where the initialization of the MPI environment has to be carried out firstly [18]. We then perform an Arnoldi factorization, and replicate matrix $\mathbf{H}$ on each node. The next step is to partition the calculation of $\mathbf{V}$ matrix by the available number of processors, and distributes it to

each node. Each processor performs the respective matrix operation, exchanges data to other processors. With this approach there are two communication points within the construction of the Arnoldi factorization: the computation of the 2-norm of the distributed vector f_m and the orthogonalization of f_m to $\mathbf{V_m}$ using classical Gram–Schmidt with DGKS correction [6,25,26]. Typically the partitioning of $\mathbf{V}$ is comparable with the parallel decomposition of the matrix $\mathbf{A}$. For n-node PCs cluster, the $\mathbf{V}$ matrix is partitioned by blocks $\mathbf{V^T} = (\mathbf{V^{(1)T}}, \mathbf{V^{(2)T}}, \ldots, \mathbf{V^{(n)T}})$ with one block per processor and with $\mathbf{H}$ replicated on each node. Since $\mathbf{H}$ is replicated on each processor, all operations on the matrix $\mathbf{H}$ are replicated on each node and there is no communication overhead. We note that the parallel technique is that the partition of operations on the matrix can accelerate the calculations and reduce communications among processors.

3. Results and discussion

We first compare the convergence properties of the used three methods, shown in Table 2. The QR is direct method which enables us to compute all eigenvales (if the target is for all eigenvalues); however, the implicitly restarted Arnoldi (IR-Arnoldi) and Jacobi–Davidson (J–D) methods belong to iterative methods. Therefore, we merely compute first few eigenvalues, target $= 50$. It is found for $L < 10$, the QR method solves all targets acceptable. Unfortunately, it takes a long time to solve large-scale problems (e.g., 5 h for an 11-mer on a single PC). Only a half of the target converges when the J–D method is adopted. It is known that the J–D method is an empirical technique, so its convergence is still an ambiguous problem [27,23].

As shown in Table 3, we perform a convergence test on the IR-Arnoldi and J–D methods for a problem with 11-mer, where four different sequences are calculated. We find the convergence property of the J–D method depends on the combination of sequence. Using the IR-Arnoldi method, we estimate the CPU time for solving the first 10 eigenvalues of the problem with the 11-mer hydrophobic residues only (the simplest case), as shown in Table 4. For different sequence types, the CPU time increase from 3 to 8 times of this problem.

As shown in Fig. 2, a sequence with nine residues which has only one conformation in ground state is solved with the QR method, where all eigenvalues of the problem are computed. It is found that the largest computed nonpositive eigenvalue is equal to -0.0422. The folding time is proportional to the reciprocal of -0.042. Shown in Table 5, we have tested the stability of the IR-Arnoldi method with different initial

Table 4
A CPU-time test on the IR-Arnoldi method for the problem with 11-mer hydrophobic residues only, where the length of sequences are from 12 to 17 on the same PC

Length of sequence	CPU time (s.)
4	5
12	12
13	39
14	115
15	343
16	906
17	2880

Table 2
The interested target and converged result of the tested problem (we consider here the hydrophobic residues only) with different methods

	Matrix size	$(740)^2$ (9-mer)	$(2034)^2$ (10-mer)	$(5513)^2$ (11-mer)	$(15,037)^2$ (12-mer)
IR-Arnodi	Target	50	50	50	50
	Converged	50	50	50	50
J–D	Target	50	50	50	50
	Converged	26	24	21	20
QR	Target	740	2034	5513	15,037
	Converged	740	2034	5513	N/A

N/A means that the memory size is greater than 2 GB on the used single PC with 3.06 GHz CPU.

Table 3
A convergence test on the IR-Arnoldi and J–D methods for a problem with 11-mer, where four sequences are calculated

Sequence	IR-Arnoldi		J–D methods	
	Target	Converged	Target	Converged
11111111111	50	50	50	21
10101010101	50	50	50	21
00111101011	50	50	50	19
11001111011	50	50	50	20

Solution with the QR method demands on huge computing resource. We compare only two methods on the same PC. The CPU time of these two methods is within 4 h.

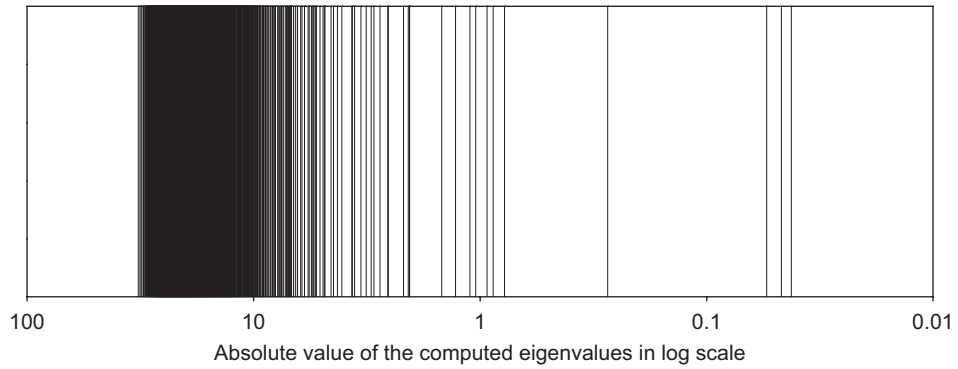

Fig. 2. The distribution of all computed eigenvalues using the QR algorithm. The plot is for the absolute value of the computed eigenvalues. The first three largest nonpositive eigenvalues are −0.0422, −0.0468, and −0.0542, respectively. They determine the smallest mode of folding.

Table 5
A stability test of the IR-Arnoldi method with different initial guesses for the protein with nine residues

Initial guess	The 1st eigenvalue	The 2nd eigenvalue
1	0	−0.04223398667659
0.04	0	−0.04223398667659
0.0001	0	−0.04223398667664
0	0	0.01827176116506
−0.001	0	−0.04223398667660
−0.04	−0.04223398667659	−0.04675639270119
−1	−1.04365630038126	−0.93086218558657

guesses. We find the IR-Arnoldi method is sensitive to the initial guesses which may complicate the solution of master equation. Besides the zero initial guess, biological-based observations could also be adopted as initial guesses to reduce the related side effects.

To verify the stability and correctness of the parallel computational method, we test the sequence 0011101011 with 10 residues which has unique conformation in the ground state. With similar PC-based Linux cluster to [18,19], each PC is equipped IBM eServer EM64T with 3.6 GHz CPU, 2 GB memory, and Intel 100 MBit fast Ethernet. All PCs in the constructed 32-nodes cluster system are connected with 100 MBit 3Com Ethernet switch. To verify the correctness of the parallel solution technique, we plot the distributions of all computed eigenvalues, shown in Fig. 3, and Table 6 shows the first ten computed eigenvalues from a single processor PC and 32-node PC-based Linux cluster. Obviously, the computed eigenvalues are totally the same in a single PC and 32-node PC-based cluster [17]. We also perform other verifications with larger sequences and have similar results. This indicates that the implemented parallel method provides a computationally efficient way to accurately calculate the eigenvalues in studying dynamics of protein folding. Fig. 4 shows the maximum norm error of the smallest and 50th eigenvalue versus the number of iterations in a single PC and 32-node PC-based cluster. The convergence of the smallest eigenvalue is much faster than that of the fiftieth eigenvalue. It is also found that our parallel method has a similar behavior of convergence in 32-nodes cluster compared with the behavior in a single PC.

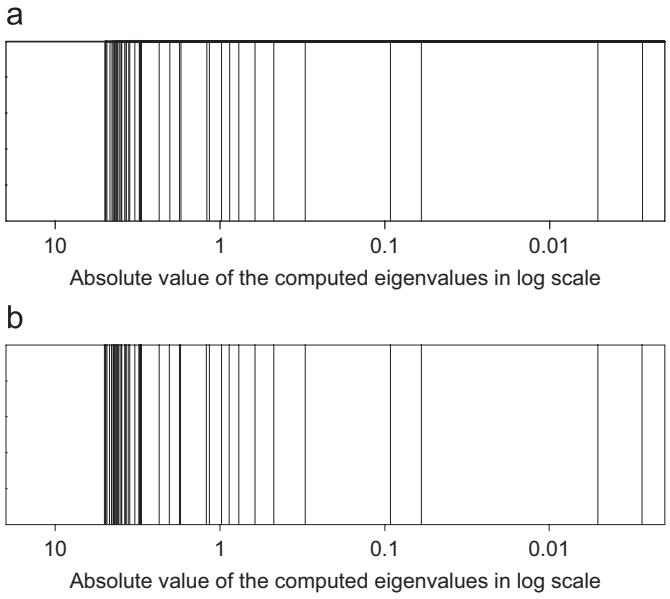

Fig. 3. The distribution of all computed eigenvalues for the sequence 0011101011 ($L = 10$) in (a) a single processor (b) and 32-node PC-based Linux cluster.

Table 6
The computed first 10 eigenvalues on a single processor and 32-node PC-based Linux cluster with a 10-mer sequence

Eigenvalues	Single processor	32-nodes cluster
1st	2.7300e − 3	2.7300e − 3
2nd	5.0800e − 3	5.0800e − 3
3rd	0.0599	0.0599
4th	0.0927	0.0927
5th	0.3039	0.3039
6th	0.4733	0.4733
7th	0.6146	0.6146
8th	0.7724	0.7724
9th	0.8789	0.8789
10th	0.9818	0.9818

The benchmarks, speedup, efficiency, and maximum difference [18,19,10,9] with respect to various numbers of processors and matrix sizes are adopted to evaluate the parallel

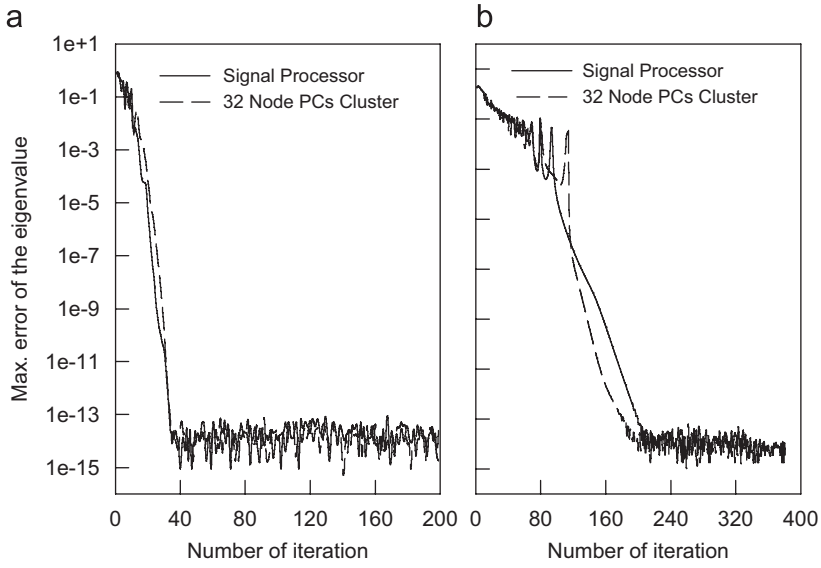

Fig. 4. The maximum norm error of the (a) first and (b) the 50th eigenvalues versus the number of iterations.

Table 7
The achieved parallel speedup and efficiency of the parallel computing algorithm, where the tested case is a 17-mer with only hydrophobic residues

Processors	Simulation time (s)	Speedup	Efficiency (%)
1	34,704	–	–
2	18,964	1.83	91.50
4	9859	3.52	88.00
8	5119	6.78	84.75
16	2902	11.96	74.75
32	1496	23.20	72.50

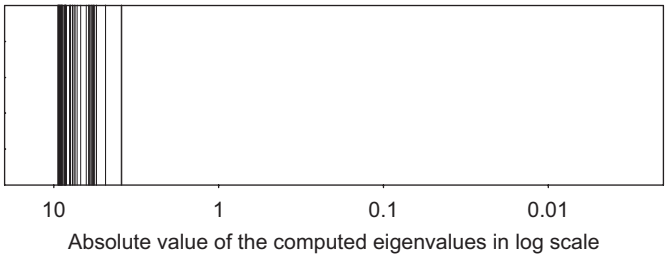

Fig. 5. The eigenvalues of the transition matrix of a 17-mer with only hydrophobic residues. The dimension of the transition matrix is 2,155,667 by 2,155,667, and 50 eigenvalues have been computed.

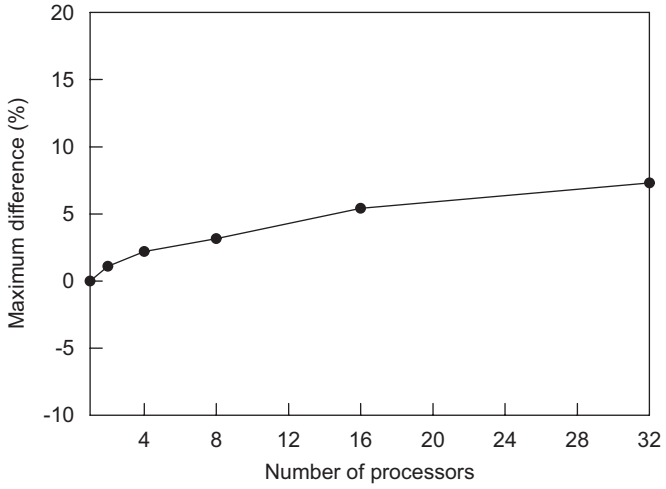

Fig. 6. The achieved load balancing versus the number of processors for the tested case of a 17-mer with only hydrophobic residues.

performances. The speedup is the ratio of the code execution time on a single processor to that on multiple processors. The efficiency is defined as the speedup divided by the number of processors. Table 7 summarizes the parallel speedup and efficiency of the implemented parallel technique. The tested cases is a 17-mer with only hydrophobic residues, and we compute the first 50 eigenvalues of the corresponding transition matrix. Fig. 5 is the computed eigenvalues of the transition matrix of a 17-mer with only hydrophobic residues. The dimension of the transition matrix is 2,155,667 by 2,155,667, and 50 eigenvalues have been computed. We find that the number of processors increases, and the efficiency decreases, as shown in Table 7. However, a 23-times speedup is maintained and the efficiency is over 72% on the 32-nodes cluster for the 17-mer case. For the same tested case, as shown in Fig. 6, the variation of maximum difference is within 8% when the number of processors is increased from 2 to 32. The maximum difference is defined as the maximum difference of the code execution time divided by the maximum execution time [18,19,10,9]. Fig. 7 shows the achieved parallel efficiency versus the matrix size for different number of processors. The parallel efficiency can hold over 70% for parallelization on 2, 4, and 8 processors, which is almost independent of matrix size. It even achieves 90% efficiency on 2 processors for all matrix sizes. The parallelization on 16 and 32 processors shows the efficiency increases

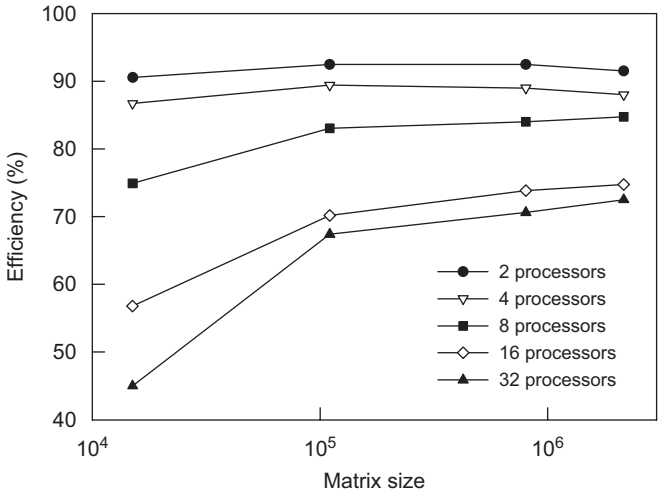

Fig. 7. The achieved parallel efficiency versus the matrix size for different number of processors.

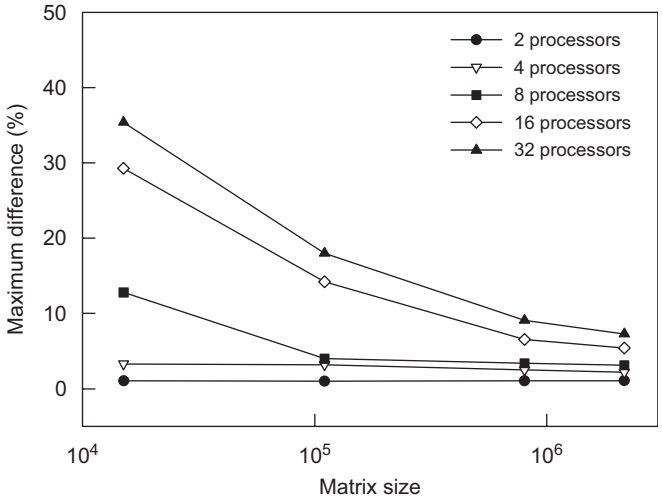

Fig. 8. The achieved load balancing versus the matrix size for different number of processors.

when the matrix size increase. However, the efficiency is degraded for the cases with small matrix size due to the increased data communication time among the processors. According to the results, when the matrix size is less than 10^5, it is suitable to perform parallelization on the small number of processors (e.g., < 8) for maintaining optimal speedup and efficiency. As shown in Fig. 7, for the parallelization on 16 processors, the slope of efficiency is reduced when the matrix size is greater than 10^6, whereas the slope of efficiency of the parallelization on 32 processors still grows when the matrix size increases. The results imply that we may perform parallelization on 32 or more than 32 processors to gain an improved speedup without losing too much parallel efficiency when the matrix size is greater than 10^6. Fig. 8 shows the maximum difference versus the matrix size for different number of processors. We notice that the maximum differences of the parallelization on 16 and 32 processors decrease significantly when the matrix size increases. We thus have an opportunity to achieve a better load balancing for those cases with large matrix size. This observation has been confirmed for the parallelization of the tested case of a 17-mer with only hydrophobic residues on 32 processors (where the matrix size is 2,155,667 by 2,155,667), as shown in Figs. 6 and 8.

4. Conclusions

We have computationally studied the dynamics of protein folding by directly solving a large-scale matrix eigenvalue problem with three numerical algorithms, the implicitly restarted Arnoldi, Jacobi–Davidson, and QR methods. The QR method demands huge computing resource when the length of sequence $L > 10$. The Jacobi–Davidson method has encountered convergent problems, for cases of $L > 9$. The implicitly restarted-Arnoldi method solves different corresponding matrix problems among three methods. We have further parallelized the implicitly restarted-Arnoldi method for accelerating the solution of large-scale eigenvalue problem on our 32-nodes PC-based Linux cluster. Accuracy of the parallelization of implicitly restarted Arnoldi method has been confirmed for several tested cases. Benchmark results including the speedup and parallel efficiency have also been achieved, and have exhibited excellent parallel performance on the PC cluster. We believe this approach enables us to solve large-scale eigenvalue problems of the master equation in dynamics of protein folding. For problems with more large sequences, we note that the growth of matrix size is very fast and then ways to construct the corresponding matrix should be explored. Advanced computational techniques, such as distributed memory scheme among computational nodes could also benefit the solution of large-scale problems. We are currently planning to upgrade our cluster with more number of processors for further large-scale problems.

Acknowledgements

This work was supported in part by the National Science Council of TAIWAN under Contract NSC-96-2221-E-009-210, Contract NSC-95-2221-E-009-336, Contract NSC-96-2752-E-009-003-PAE, Contract NSC-95-2752-E-009-003-PAE, and by the MoE ATU Program, Taiwan, under a 2006–2007 grant.

References

[1] M. Akahoshi, K. Onizuka, M. Ishikawa, K. Asai, A three-dimensional animation system for protein folding simulation, in: Proceedings of 27th International Conference on Biotechnology Computing 5 (1994) 173–182.
[2] E. Alm, A.M. AV, T. Kortemme, D. Baker, Simple physical models connect theory and experiment in protein folding kinetics, J. Mol. Biol. 322 (2002) 463–476.
[3] S. Altmeyer, R. Fuchslin, J. McCaskill, Folding stabilizes the evolution of catalysts, Artificial Life 10 (2004) 23–38.
[4] G. Bologna, R. Appel, A comparison study on protein fold recognition, in: Proceedings of the 9th International Conference on Neural Information, vol. 5, 2002, pp. 2492–2496.
[5] V. Daggett, Molecular dynamics simulations of the protein unfolding/folding process, Accounts of Chem. Res. 35 (2002) 422–430.

[6] J. Daniel, W. Gragg, L. Kaufman, G. Stewart, Reorthogonalization and stable algorithms for updating the Gram-Schmidt qr factorization, Math. Comput. 30 (1976) 772–795.
[7] K. Dill, H. Chan, From levinthal to pathways to funnels, Nat. Structural Biol. 4 (1997) 10–18.
[8] K.A. Dill, S. Bromberg, K. Yue, K.M. Fiebig, D.P. Yee, P.D. Thomas, H.S. Chan, Principles of protein folding—a perspective from simple exact models, Protein Sci. 4 (1995) 561–602.
[9] K. Dowd, C. Severance, High Performance Computing, O'Reilly, Sebastopol, 1998.
[10] H. El-Rewini, T.G. Lewis, Distributed and Parallel Computing, Manning, Greenwich, CT, 1998.
[11] S.W. Englander, Protein folding intermediates and pathways studied by hydrogen exchange, Annual Rev. Biophy. Biomol. Struct. 29 (2000) 213–238.
[12] D.R. Fokkema, G.L.G. Sleijpen, H.A.V. Vorst, Jacobi–Davidson style qr and qz algorithms for the reduction of matrix pencils, SIAM J. Sci. Comput. 20 (1998) 94.
[13] U. Hansmann, Protein folding in silico: an overview, Comput. Sci. Engrg. 5 (2003) 64–69.
[14] C.-D. Huang, C.-T. Lin, N.R. Pal, Hierarchical learning architecture with automatic feature selection for multiclass protein fold classification, IEEE Trans. NanoBiol. 2 (2003) 221–232.
[15] E.S. Keum, J. Kim, K.J. Santos, Local minima-based exploration of off-lattice protein folding, in: Proceedings of the 2003 IEEE Bioinformatics Conference, 2003, pp. 615–616.
[16] R.B. Lehoucq, D. Sorensen, Deflation techniques for an implicitly re-started Arnoldi iteration, SIAM J. Matrix Anal. Appl. 17 (1996) 789–821.
[17] Y. Li, Numerical calculation of electronic structure for three-dimensional nanoscale semiconductor quantum dots and rings, J. Comput. Electron. 2 (2003) 49–57.
[18] Y. Li, S. Sze, T. Chao, A practical implementation of parallel dynamic load balancing for adaptive computing in VLSI device simulation, Engr. Comput. 18 (2002) 124–137.
[19] Y. Li, S. Yu, A two-dimensional quantum transport simulation of nanoscale double-gate MOSFET's using parallel adaptive technique, IEICE Trans. Info. Syst. E87-D (2004) 1751–1758.
[20] J.F.-B.R.G.M.A. Micheelsen, C. Rischel, L. Serrano, Mean first-passage time analysis reveals rate-limiting steps, parallel pathways and dead ends in a simple model of protein folding, Europhy. Lett. 61 (2003) 561–566.
[21] V. Munoz, W.A. Eaton, A simple model for calculating the kinetics of protein folding from three-dimensional structures, Proc. Nat. Acad. Sci. U.S.A. 96 (1999) 11311–11316.
[22] K.T.S.K.W. Plaxco, D. Baker, Contact order, transition state placement and the refolding rates of single domain proteins, J. Mol. Biol. 277 (1998) 985–994.
[23] G.L.G. Sleijpen, H.A.V. Vorst, A Jacobi–Davidson iteration method for linear eigenvalue problems, SIAM J. Matrix. Anal. Appl. 42 (1996) 267–293.
[24] D.C. Sorensen, Implicit application of polynomial filters in a k-step Arnoldi method, SIAM J. Matrix Anal. Appl. 13 (1992) 357–385.
[25] D.C. Sorensen, Implicitly-restarted arnoldi/lanczos methods for large scale eigenvalue calculations, in: D.E. Keyes, A. Sameh, V. Venkatakrishnan (Eds.), Parallel Number Alogirthm Proceedings ICASE/LaRC Workshop, Kluwer Academic Publishers, Dordrecht, 1995.
[26] A. Stathopoulos, K. Wu, A block orthogonalization procedure with constant synchronization requirements, SIAM J. Sci. Comput. 23 (2002) 2165–2182.
[27] A. Stathopoulos, S. Yousef, W. Kesheng, Dynamic thick restarting of the Davidson, and the implicity restarted Arnoldi methods, SIAM J. Sci. Comput. 19 (1998) 227–245.
[28] D.S. Watkins, Qr-like algorithms for eigenvalue problems, J. Comput. Appl. Math. 123 (2000) 67–83.

Yiming Li received his B.S. degrees in applied mathematics and electronics engineering, his M.S. degree in applied mathematics, and his Ph.D. degree in electronics from the National Chiao Tung University (NCTU), Hsinchu, Taiwan, in 1996, 1998, and 2001, respectively. He is currently an Associate Professor with the Department of Communication Engineering, NCTU. He is a Deputy Director of the Modeling and Simulation Center and conducts the Parallel and Scientific Computing Laboratory at NCTU. His research areas include computational science and engineering, in particular, for biology, electronics and physics. He has authored over 120 research papers appearing in international book chapters, journals, and conferences. He has organized and served on various international conferences and has served as a reviewer and editor for many international journals. Dr. Li is a member of Phi Tau Phi, Sigma Xi, ACM and IEEE, and is included in *Who's Who in the World*. He was the recipient of the 2002 Research Fellowship Award presented by the Pan Wen-Yuan Foundation, Taiwan and the 2006 Outstanding Young Electrical Engineer Award from Chinese Institute of Electrical Engineering, Taiwan.

Shao-Ming Yu received his B.S. and M.S. degrees in Computer and Information Science from NCTU in 2002 and 2004. Currently he is pursuing his Ph.D. degree at the Department of Computer Science of NCTU. His research interests focus on modeling and simulation of semiconductor nanodevices, parallel and scientific computation, evolutionary algorithms, and design optimization. He is a student member of IEEE.

Yih-Lang Li received his B.S. degree in nuclear engineering and his M.S. and his Ph.D. degrees in computer science from the National Tsing Hua University, Hsinchu, Taiwan, in 1987, 1990, and 1996, respectively. In February 2003, he joined the faculty of the Department of Computer Science, NCTU, where he is currently an Assistant Professor. Prior to joining the faculty of NCTU, from 1995 to 1996 and from 1998 to 2003, he was a Software Engineer and an Associate Manager at Springsoft Corporation, Hsinchu, where he was heavily involved in the development of verification and synthesis tools for custom-based layout. His research interests include physical synthesis, parallel architecture, and VLSI testing.