Decentralized Replica Exchange Parallel Tempering: An Efficient Implementation of Parallel Tempering Using MPI and SPRNG

Yaohang Li[1], Michael Mascagni[2], and Andrey Gorin[3]

[1] Department of Computer Science
North Carolina A&T State University, Greensboro, NC 27411
yaohang@ncat.edu
[2] Department of Computer Science
Florida State University, Tallahassee, FL 32306
mascagni@cs.fsu.edu
[3] Division of Computer Science and Mathematics
Oak Ridge National Laboratory, Oak Ridge, TN 37831
agor@ornl.gov

Abstract. Parallel Tempering (PT), also known as Replica Exchange, is a powerful Markov Chain Monte Carlo sampling approach which aims at reducing the relaxation time in simulations of physical systems. In this paper, we present a novel implementation of PT, so-called decentralized replica exchange PT, using MPI and the Scalable Parallel Random Number Generators (SPRNG) libraries. By adjusting the replica exchange operations in the original PT algorithm, and taking advantage of the characteristics of pseudorandom number generators, this implementation minimizes the overhead caused by interprocessor communication in replica exchange in PT. This enables one to efficiently apply PT to large-scale massively parallel systems. The efficiency of this implementation has been demonstrated in the context of various benchmark energy functions, such as the high-dimensional Rosenbrock function, and a rugged funnel-like function.

Keywords: Parallel Tempering, Monte Carlo Methods, Parallel Programming.

1 Introduction

Parallel Tempering (PT), also known as Replica Exchange or the Multi-Markov Chain method, is a powerful Markov Chain Monte Carlo (MCMC) sampling scheme proposed by Geyer and Thompson [1], and Marinari and Parisi [2]. In parallel tempering, multiple independent replicas of a system are simulated simultaneously under different thermodynamic conditions, the differences defined by temperatures in most cases. Replicas at high temperature are generally capable of experiencing a larger volume of the phase space while those at low temperature are able to explore the "local detail" of the energy landscape. During the process of simulation, neighboring replicas are allowed to exchange configurations from time to time, subject to the acceptance criterion. By carefully setting up the temperature ladder and the number of replicas, PT can reduce the relaxation time of the Monte Carlo simulations in the physical systems, and improve convergence to a global minimum. PT is ideal for complex physical systems that are characterized by rough energy landscapes. Successful PT applications include the simulation of biomolecules [3], determination of X-ray structures [4], polymers [5], and structure prediction in small proteins [6], [7].

Intuitively, PT simulation is a natural fit for parallel computing systems because multiple replicas are allowed to run simultaneously at different temperatures. Each replica simulation can be realized as an independent process running on its own CPU. However, replica exchange operations in PT can become computationally expensive for large-scale simulations, due to the number of replicas needed as well as the interprocessor communication overhead between replicas.

In this paper, we present our novel decentralized replica exchange parallel tempering implementation. Our implementation is based on the MPI and SPRNG (Scalable Parallel Random Number Generators) [8] libraries. Functions in the MPI library are used for necessary interprocessor communication in the parallel computing environment. The SPRNG library provides parameterized pseudorandom number generators to produce independent random number streams for parallel processes. By taking advantage of the determinism and reproducibility characteristics of pseudorandom number streams, distributed processes can come to a common decision without performing interprocessor communication. Moreover, temperature exchange instead of configuration exchange is used to reduce the amount of communication in replica exchange. To eliminate the additional global synchronization posed by temperature exchange, we extend the neighboring replica exchange in the original PT scheme to a more generalized random replica exchange. All these efforts lead to a decentralized implementation of replica exchange transitions in PT, and thus minimize the interprocessor communication overhead in parallel PT applications.

2 The Parallel Tempering Scheme

In a general, the PT algorithm using MCMC for local sampling works as follows. A composite system with N sets of replicas is constructed with one replica per temperature level, T_i. Multiple temperature levels form a temperature ladder. A state of the composite system is specified by $X = \{x_1, x_2, \ldots, x_N\}$, where x_i is the replica at temperature level i. The equilibrium distribution of the composite system, X, is,

$$\Pi(X) = \prod_{i=1}^{N} \frac{e^{-\beta_i E(x_i)}}{Z(T_i)}, \tag{1}$$

where $\beta_i = 1/T_i$, $E(x_i)$ is the energy function, and $Z(T_i) = \int e^{-\beta_i E(x_i)} dx_i$, is the partition function of the replica at T_i.

At each iteration step, t, the Markov chains can be realized with two types of transitions – the Metropolis transition and the replica transition:

1. Metropolis Transition: The Metropolis transition is employed for local Monte Carlo moves for the conformation at each temperature level. The transition probability only depends on the change of in the objective function, $E(x_i)$, where x_i is

the conformation at temperature level T_i. A new configuration x_i' is sampled from the proposal distribution $q_i(.|x_i)$. The Metropolis-Hastings ratio at temperature level T_i is calculated as:

$$w_{Local}(x_i \to x_i') = e^{-\beta_i \Delta_i E} = e^{-\beta_i (E(x_i') - E(x_i))}, \tag{2}$$

The new state is accepted with the probability $\min(1, w_{Local}(x_i \to x_i'))$. The detailed balance condition holds for each replica in Metropolis transition and therefore, it also holds for the composite system.

2. Replica Transition: The replica transition takes place with the probability θ and is used to exchange conformations at two neighboring temperature levels, i and $i+1$.

$$x_i \leftrightarrow x_{i+1}. \tag{3}$$

The exchange is accepted according to the Metropolis-Hastings criterion with probability

$$\begin{aligned} &P_{\mathrm{Re}\,plica}(x_i \leftrightarrow x_{i+1}) = P(\{x_1,...,x_i,x_{i+1},...,x_N\} \mid \{x_1,...,x_{i+1},x_i,...,x_N\}) \\ &= \min(1, \frac{\Pi(\{x_1,...,x_{i+1},x_i,...,x_N\})}{\Pi(\{x_1,...,x_i,x_{i+1},...,x_N\})}) \\ &= \min(1, e^{-\beta_i E(x_{i+1}) - \beta_{i+1} E(x_i) + \beta_{i+1} E(x_{i+1}) + \beta_i E(x_i)}) \end{aligned} \tag{4}$$

The relaxation rate [9] can be characterized by the ergodic measure via the so-called fluctuation metric,

$$\Omega(X^t) = \sum_{j=1}^{N} [\beta_j E(x_j^t) - \overline{\beta E(x^t)}]^2 / N, \tag{5}$$

where $\overline{\beta E(x^t)} = \sum_{k=1}^{N} \beta_k E(x_k{}^t) / N$ is the ergodic average at iteration step t. The replica transitions lead to an improvement of the relaxation rate of the overall simulation of the composite system. Using the definition of the replica exchange probability, the detailed balance equation can be obtained for replica transition.

$$\begin{aligned} &P(\{x_1,...,x_i,x_{i+1},...,x_N\} \mid \{x_1,...,x_{i+1},x_i,...,x_N\}) \Pi(\{x_1,...,x_{i+1},x_i,...,x_N\}) \\ &= P(\{x_1,...,x_{i+1},x_i,...,x_N\} \mid \{x_1,...,x_i,x_{i+1},...,x_N\}) \Pi(\{x_1,...,x_i,x_{i+1},...,x_N\}) \end{aligned} \tag{6}$$

Descriptive pseudo code of the PT algorithm follows.

```
Initialize $N$ replica $x_1$, $x_2$, …, $x_N$ and their corre-
sponding temperatures $T_1$, $T_2$, …, $T_N$
Initialize t ← 0
Repeat {
// Perform Metropolis Transition
for each replica $i$ {
   Sample a point $x_i'$ from $q_i( . | x_i )$
   Sample a uniform [0, 1) random variable $U_M$
```

```
      if $U_M$ <= $w_{local}(x_i \rightarrow x_i')$ then $x_i \leftarrow x_i'$
      }
   //Perform Replica Transition
   Sample a uniform [0, 1) random variable $U_R$
   if $U_R$ <= $\theta$ then {
      Sample an integer variable $i$ from U[1, $N-1$]
      Sample a uniform [0, 1) random variable $U_S$
      if  $U_S$ <= $P_{Replica}(x_i \leftrightarrow x_{i+1})$ then
              $x_i \leftrightarrow x_{i+1}$
      }
   Increment t
   }
```

3 Decentralized Parallel Implementation

3.1 Pseudorandom Number Reproducibility for Global Process Synchronization

In PT algorithms, a common decision has to be made among multiple processes to determine whether the replica transition should occur. The common decision is based on a uniform [0, 1) random number. Instead of producing a uniform pseudorandom number and then broadcasting it to other processes, a clever implementation is to use a random number generator with the same parameters and seed, for the replica transition decision in each individual process. A pseudorandom number generator is deterministic and reproducible, i.e., with the same parameters and seed, the generator will always produce the identical random number stream. Taking advantage of the reproducibility characteristic of good pseudorandom number generators, distributed processes can come to a common decision without global process synchronization. Similarly, the common decisions in which two processes will participate in replica exchange and whether the replica exchange attempt will be accepted can be made by using the same random number streams in multiple processes without communication among processes.

In our parallel implementation of the PT algorithm, multiple random number streams are used to minimize interprocessor communication; however, the problem of possible correlation among the random number streams arises. Intra-stream correlation will form sophisticated pattern, which may lead to defective or even erroneous results in Monte Carlo simulations. To avoid the intra-stream correlation problem, we employ the SPRNG library, which can produce up to 2^{78000} - 1 independent random number streams with sufficiently long period and good quality via appropriate parameterization. Properly configuring the random number generators in the SPRNG library, independence of the parallel random number streams used in a parallel PT implementation can be ensured [8], [12].

3.2 Configuration Exchange or Temperature Exchange in Replica Exchange?

Replica exchange is employed in the PT scheme for improving mixing among the Markov chains running at various temperature levels. Replica exchange requires system passing configuration information between two processes carrying out the corresponding Markov chains. In many practical simulation applications, e.g., a large protein with hundreds of residues, or a physical system with thousands of molecules, replica exchange by swapping the system configurations will be rather costly because of large amount of interprocessor communication required. An alternative way to reduce the communication is to use temperature exchange instead. Compared to configuration exchange, temperature exchange only requires swapping of the temperature T_i, energy function value $E(x_i)$, and proposal distribution function $q_i(.|.)$ for index i, if different proposal functions are used in different processes. Temperature exchange only requires swapping of at most two floating point numbers and one integer index. As a result, temperature exchange is much more communication friendly than configuration exchange in complex system simulations.

3.3 Neighboring Replica Exchange or Random Replica Exchange?

If temperature exchange is used instead of configuration exchange for our replica exchange, the amount of interprocessor communication can be significantly reduced in complex systems with large amounts of configuration information. However, the temperature order is disturbed in temperature exchange, which is no longer ordered by process rank. As a result, after several steps of temperature exchange, swapping of neighboring processes does not lead to exchange of neighboring temperature levels. Performing replica exchange at neighboring temperatures requires global awareness of the temperature distribution at different processes, which demands additional global process synchronization by gathering the temperature values distributed on different processes.

Instead of replica exchange at neighboring temperature levels, a more general form of replica exchange is random replica exchange, where replica exchange takes place between any two randomly selected temperature levels, i and j.

$$x_i \leftrightarrow x_j \,. \tag{7}$$

Neighboring replica exchange is a special case of random replica exchange where $i = j + 1$. Accordingly, the exchange is accepted according to the Metropolis-Hastings criterion with probability

$$\begin{aligned} &P_{\mathrm{Re}\,plica}(x_i \leftrightarrow x_j) = P(\{x_1,\ldots,x_i,\ldots,x_j,\ldots,x_N\} \mid \{x_1,\ldots,x_j,\ldots,x_i,\ldots,x_N\}) \\ &= \min(1, \frac{\Pi(\{x_1,\ldots,x_j,\ldots,x_i,\ldots,x_N\})}{\Pi(\{x_1,\ldots,x_i,\ldots,x_j,\ldots,x_N\})}) \\ &= \min(1, e^{-\beta_i E(x_j) - \beta_j E(x_i) + \beta_j E(x_j) + \beta_i E(x_i)}) \end{aligned} \tag{8}$$

Notice that the detailed balance condition still holds for random replica exchange transitions.

$$\begin{aligned} &P(\{x_1,\ldots,x_i,\ldots,x_j,\ldots,x_N\} \mid \{x_1,\ldots,x_j,\ldots,x_i,\ldots,x_N\})\Pi(\{x_1,\ldots,x_j,\ldots,x_i,\ldots,x_N\}) \\ &= P(\{x_1,\ldots,x_j,\ldots,x_i,\ldots,x_N\} \mid \{x_1,\ldots,x_i,\ldots,x_j,\ldots,x_N\})\Pi(\{x_1,\ldots,x_i,\ldots,x_j,\ldots,x_N\}) \end{aligned} \tag{9}$$

Two unbiased participant processes in random replica exchange can be determined by a shared global random number, where global synchronization is not necessary. The random replica exchange can be thought of as a "larger" replica transition step in PT, which allows replica exchange attempts at a larger temperature difference. However, random replica exchange will have a lower success rate compared to neighboring replica exchange. Yet, using large transition steps in combination with small transition steps usually results in reduced waiting time when a system is trapped by deep local minima in a MCMC evolution [10, 11].

3.4 Random Number Streams

Various independent SPRNG random number streams, including local streams and global streams, are involved in decision making in our parallel implementation of PT. These random number streams are shown in Table 1.

Table 1. Independent Random Number Streams and Their Roles in Decentralized PT Scheme

stream name	sharing	number	decision
proposal stream	local	N	Proposal new configuration xi' for local Metropolis transition
local acceptance stream	local	N	Acceptance of local transition according to Metropolis ratio
replica exchange stream	same in all processes	1	Whether to perform replica exchange at current time step
participant stream	same in all processes	1	Whether the current process should participate in replica exchange at this time
swap stream	same in any process pair	$N*(N-1)/2$	Acceptance of replica exchange transition according to exchange ratio

3.5 Efficient Parallel PT Implementation

Fig. 1 shows the flowchart of our decentralized replica exchange PT scheme. At the beginning, the system configuration, temperature, SPRNG random number generators, and other necessary variables are initialized in each process. In Metropolis transitions, random numbers from the proposal stream are used to produce a proposal transition and then a random number from the local acceptance stream is used to determine whether the proposal transition will be accepted. Both proposal stream and local acceptance stream are local streams which are different and independent in different processes. After a Metropolis transition, in each process, a random number from the replica exchange stream is drawn to decide whether a replica transition will be performed. Both the replica exchange stream and participant stream are globally shared, where random number sequences are exactly the same in all processes. If yes, random numbers are generated in the participant stream to determine which two processes will participate in replica exchange. The non-selected processes skip replica transition. For the two randomly selected participant processes, temperature and energy function values are exchanged via the MPI function call MPI_Sendrecv(). A random number from the swap stream which is identical in both participant processes is drawn to decide whether the replica exchange attempt will be accepted. In this parallel PT implementation, the only interprocessor communication required is that for the exchange of temperature and the energy function value.

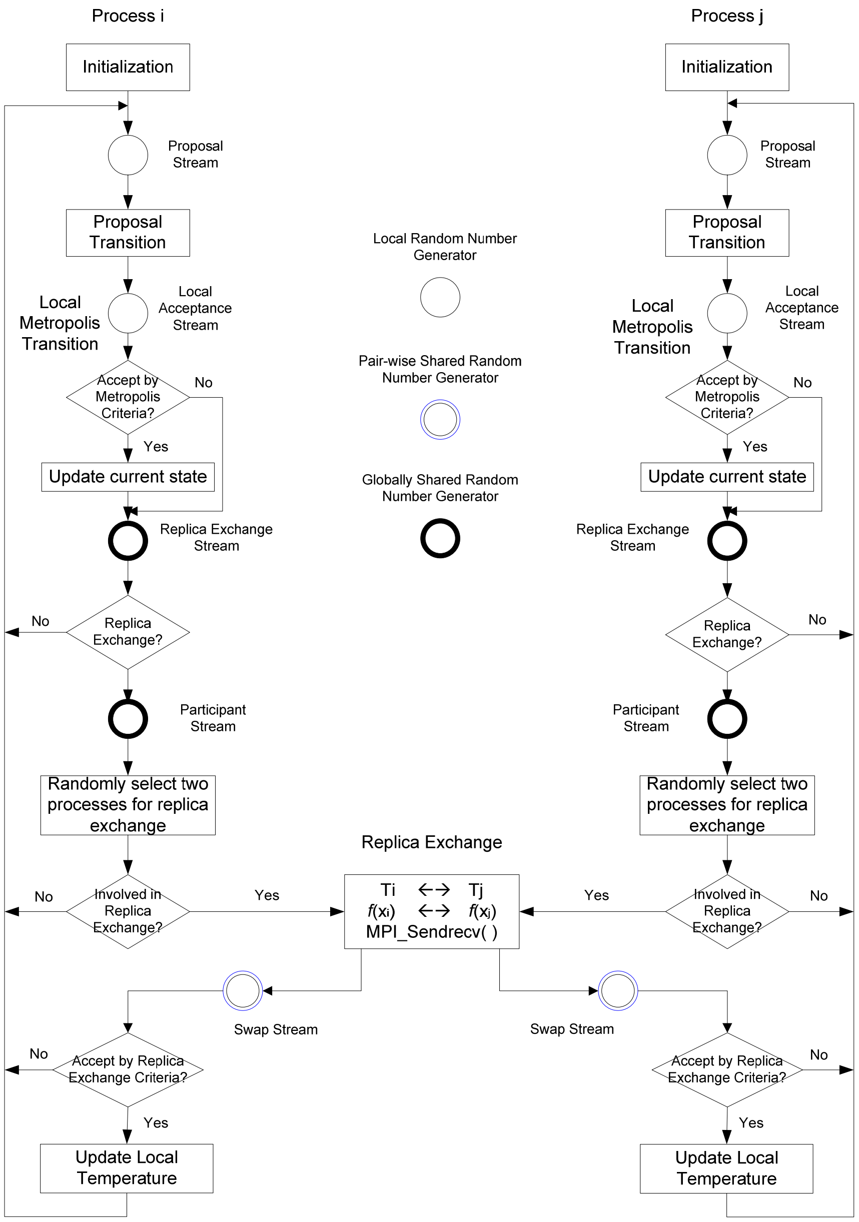
Process i
Process j
Initialization
Proposal Stream
Proposal Transition
Local Random Number Generator
Local Metropolis Transition
Local Acceptance Stream
Accept by Metropolis Criteria?
No
Yes
Pair-wise Shared Random Number Generator
Update current state
Globally Shared Random Number Generator
Replica Exchange Stream
Replica Exchange?
Participant Stream
Randomly select two processes for replica exchange
Replica Exchange
Involved in Replica Exchange?
Ti <-> Tj
f(xi) <-> f(xj)
MPI_Sendrecv( )
Swap Stream
Accept by Replica Exchange Criteria?
Update Local Temperature

Fig. 1. Flowchart of the Decentralized Replica Exchange PT Scheme

3.6 Implementation Analysis

1. Efficiency
This parallel implementation of PT eliminates global synchronization operations in PT processes. Replica exchanges, provided that they involve different processes, can be executed in parallel. The amount of communication information is also minimized by using temperature exchange and randomly choosing participant processes.

2. Reproducibility
Notice that the simulation this parallel implementation of PT is reproducible. First of all, all SPRNG random number streams involved can be exactly reproduced by retrieving the same parameters and seeds in each pseudorandom number generator. Secondly, in each process, the Metropolis transition can be reproduced by reproducing the local random numbers in the proposal stream and the local acceptance stream. Thirdly, deciding when to perform replica exchange and the participant processes are reproducible by retrieving the global random number sequences of the replica exchange stream and participant stream, respectively. Finally, when a replica exchange is attempted, each process pair can be reproduced by reproducing the corresponding swap stream.

4 Computational Results

4.1 Rosenbrock's Function

The generalized n-dimensional Rosenbrock's function is defined as

$$f(x_1,\ldots,x_n)=\sum_{i=2}^{n}(100(x_i-x_{i-1}{}^2)^2+(1-x_{i-1})^2) \quad (10)$$
,

where the global minimum is at (1.0, 1.0, …, 1.0). The Rosenbrock's function is a notorious benchmark function in optimization because of its slow convergence for

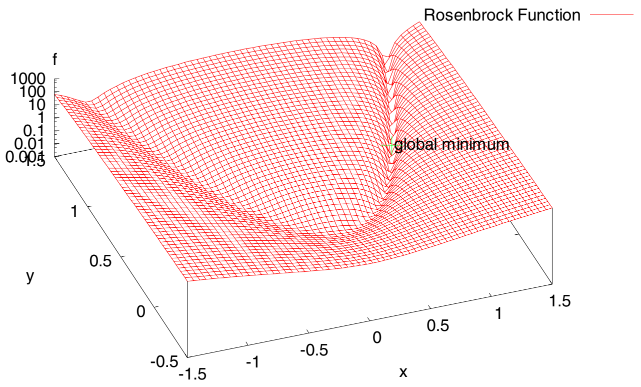

Fig. 2. Two-dimensional Rosenbrock's Function. Note the logarithmic scale of the function's axis.

most optimization methods. Due to a long narrow valley present in this function, gradient-based methods may have to spend a large number of iterations before the global minimum is reached. Fig. 2 shows the two dimensional Rosenbrock's function.

Fig. 3 shows the performance comparison of PT with decentralized replica exchange, PT using a master-slave paradigm [13], and parallel Metropolis in a 100-dimensional Rosenbrock's Function1. The replica exchange probability θ is 10%. In the master-slave parallel PT, each process carries out local Metropolis transitions and one process is designated as the master, which collects replica information from each slave process, performs replica exchange, and then scatters replica information back to each slave process. Global synchronizations are required in the master-slave paradigm. Parallel Metropolis is an naturally parallel implementation, where each process carries out a Metropolis transition, and no replica exchange takes place among the processes. From Fig. 3, one can see that the global synchronization operations in master-slave PT are costly and post heavy interprocessor communication overhead (358.3%). In contrast, the interprocessor communication overhead introduced by PT with decentralized replica exchange is small (approximately 10.2%) compared to that of the naturally parallel Metropolis in the Rosenbrock's function experiment. The resulting curves of the best, worst, and average objective function values over the number of iterations in 10 independent parallel Metropolis and PT runs are shown in Fig. 4. In this experiment, parallel Metropolis has the same initial position, temperature, and transition step size configuration as PT but does not carry out replication exchange between temperature levels. One can observe that PT exhibits a faster convergence to global minimum compared to parallel Metropolis due to reduction of relaxation time by the replication exchange transitions.

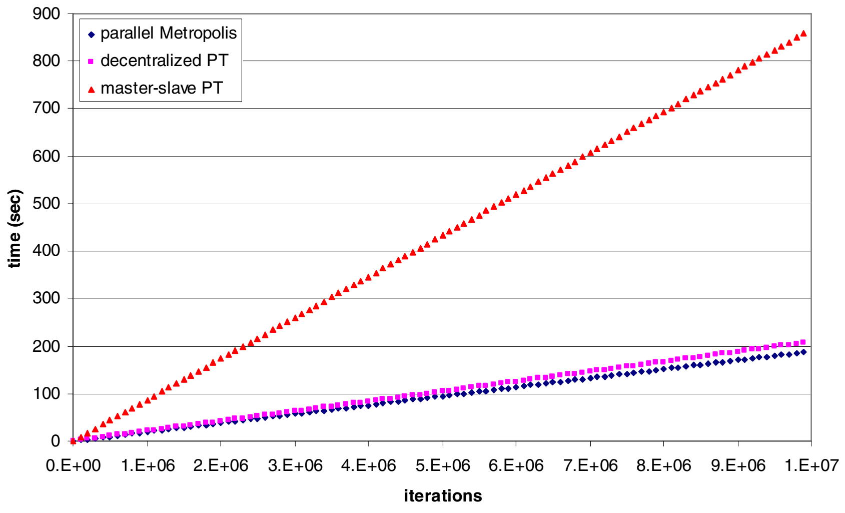

Fig. 3. Performance Comparison of Decentralized Parallel PT, Parallel Metropolis, and Master-Slave Parallel PT in Rosenbrock's Function Optimization on 8 processors

[1] The computations are carried out on a Beowulf Linux cluster with 8 2.2GHZ Xeon processors, with 1G Memory each node.

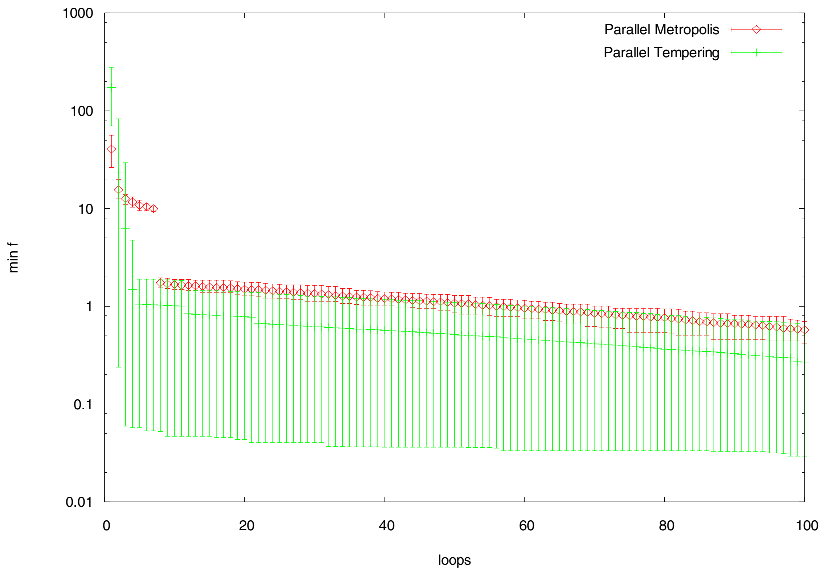

Fig. 4. Experimental Runs of Parallel Metropolis and Parallel Tempering on the 100-Dimensitonal Rosenbrock's Function. Each loop includes 10^5 iterations and shows the best, worst, and average function values.

4.2 Rugged Funnel-Like Function

In this experiment, we construct a "rugged" funnel-like function

$$E(x_1,\dots,x_n) = -\frac{c}{2n}\sum_{k=1}^{n}\left\{1 + A\sum_{i=0}^{m} a^i \cos(b^i x_k)\right\}, \qquad (11)$$

where $A = (a\text{-}1)/(a^{m+1}\text{-}1)$, n is the dimension, and m, a, b, and c are some tunable constants to determine the depth of the funnel and the number of local minima along it.

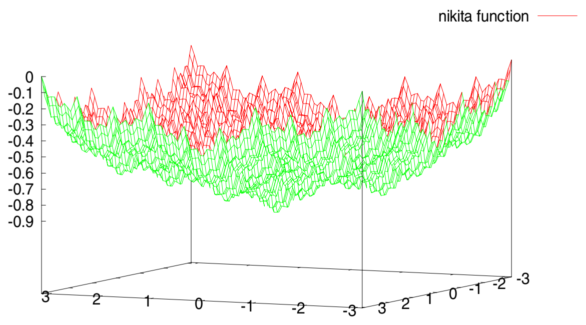

Fig. 5. A two-dimensional "Rugged" Funnel-like Function ($m = 4$, $a = 0.7$, $b = 3.0$, $c = 1.0$)

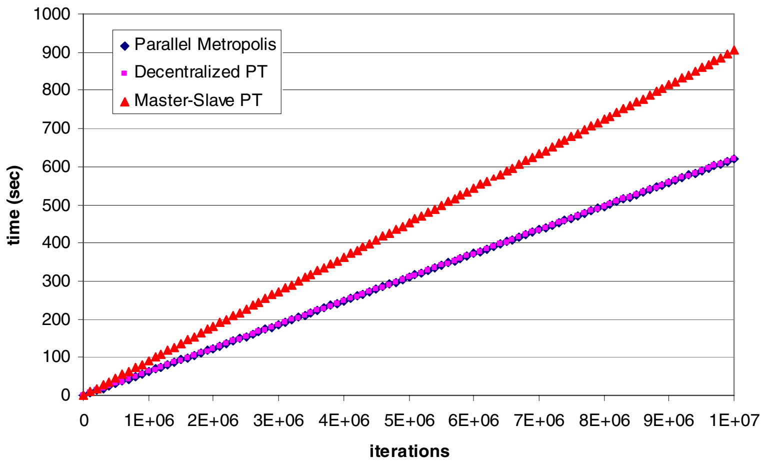

Fig. 6. Figure 6: Performance Comparison of Decentralized Parallel PT, Parallel Metropolis, and Master-Slave Parallel PT in Rugged Funnel-like Function Optimization on 8 processors

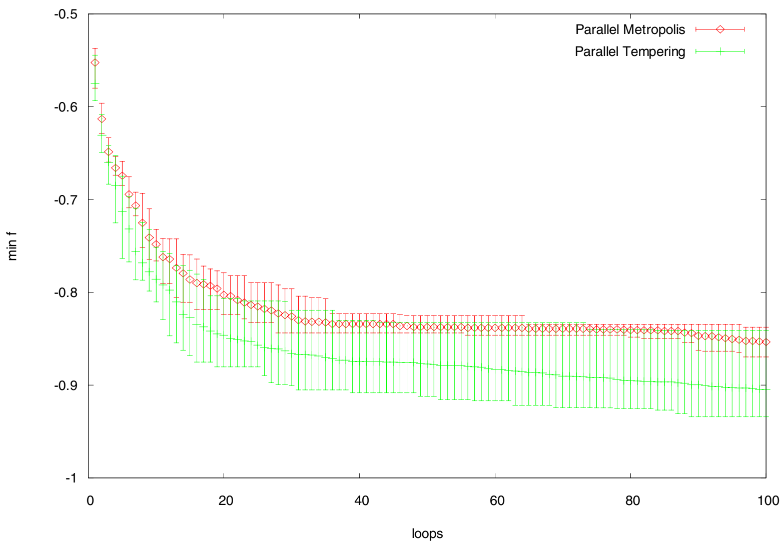

Fig. 7. Experimental Runs of Parallel Metropolis and Parallel Tempering on a 100-Dimensitonal Rugged Funnel-like Function. Each loop includes 105 iterations and shows the best, worst, and average function values.

The only global minimum is located at (0, …, 0) and is equal to -*c*. Fig. 5 shows the two-dimensional "rugged" funnel-like function where *c* = 1.0.

Fig. 6 shows the performance comparison of master-slave PT, PT with decentralized replica exchange and parallel Metropolis in a 100-dimensional rugged funnel-like function. Similar to the computational experiments on the Rosenbrock's function, decentralized replica exchange PT yields almost indistinguishable interprocessor communication overhead (0.35%) compared to parallel Metropolis and outperforms master-slave PT. As the curves of the best, worst, and average objective function values over the number of iterations in 10 independent runs shown in Fig. 7, PT exhibits a faster convergence to global minimum (0.0) than parallel Metropolis.

Notice that PT is an effective sampling method, which may locate the valley leading to the global minimum but may not be able to actually approach global minimum in high precision. Extended MCMC algorithm, such as the hybrid PT/SA algorithm [14] or combining MCMC with local minimization (downhill) methods [15], can more aggressively minimize the solution.

5 Summary

In this article, we developed a decentralized PT implementation, using the MPI and SPRNG libraries. Taking advantage of the determinism and reproducibility characteristics of parallel pseudorandom number streams in SPRNG, and using temperature exchange instead of replica exchange, we are able to eliminate the need for global synchronization and to minimize interprocessor communication. Our computational experiments, based on applying the decentralized PT implementation to the high-dimensional Rosenbrock's function and rugged funnel-like function show that insignificant amount of interprocessor communication overhead contributed to the overall simulation time. Since this decentralized PT implementation can also be applied to some extended PT algorithms, such as hybrid Parallel Tempering/Simulated Tempering [14], adaptive PT [4], parallel sintering [17], and various Evolutionary Markov Chain Monte Carlo methods [16], [18], [19], [20], this seems like a likely avenue for future work.

Acknowledgments. Yaohang Li acknowledges support by the Ralph E. Powe Junior Faculty Enhancement Award of Oak Ridge Associated Universities. The research was also sponsored by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory (ORNL), managed by UT-Battelle, LLC for the U. S. Department of Energy under Contract No. DE-AC05-00OR22725

References

1. Marinari, E., Parisi, G.: Simulated Tempering: a New Monte Carlo Scheme. Europhysics Letters 19, 451–458 (1992)
2. Geyer, C.J., Thompson, E.A.: Annealing Markov Chain Monte Carlo with Applications to Ancestral Inference. Journal of the American Statistical Association 90, 909–920 (1995)
3. Falcioni, M., Deem, M.W.: A Biased Monte Carlo Scheme for Zeolite Structure Solution. J. Chem. Phys 110, 1754–1766 (1999)
4. Schug, A., Herges, T., Verma, A., Wenzel, W.: Investigation of the parallel tempering method for protein folding. J. Phys: Condens. Matter 17, 1641–1650 (2005)
5. Sikorski, A.: Properties of Star-Branched Polymer Chains – Application of the Replica Exchange Monte Carlo Method. Macromolecules 35(18), 7132–7137 (2002)

6. Schug, A., Wenzel, W.: Predictive in-silico all atom folding of a four helix protein with a free energy model. J. Am. Chem. Soc. 126, 16737 (2004)
7. Li, Y., Strauss, C.E.M., Gorin, A.: Parallel Tempering in Rosetta Practice. In: Zhang, D., Jain, A.K. (eds.) ICBA 2004. LNCS, vol. 3072, Springer, Heidelberg (2004)
8. Mascagni, M., Srinivasan, A.: Algorithm 806: SPRNG: A Scalable Library for Pseudorandom Number Generation. ACM Transactions on Mathematical Software 26, 436–461 (2000)
9. Bu, L., Straub, J.E.: Simulating Vibrational Energy Flow in Proteins: Relaxation Rate and Mechanism for Heme Cooling in Cytochrome c. J. Phys. Chem. B 107, 12339–12345 (2003)
10. Li, Y., Protopopescu, V.A., Gorin, A.: Accelerated Simulated Tempering. Physics Letters A 328(4), 274–283 (2004)
11. Liu, J.S., Liang, F., Wong, W.H.: The Use of Multiple-Try Method and Local Optimization in Metropolis Sampling. Technical Report, Department of Statistics, Stanford University (1998)
12. Srinivasan, A., Mascagni, M., Ceperley, D.: Testing Parallel Random Number Generators. Parallel Computing 29, 69–94 (2003)
13. Li, Y., Clark, J., Zhang, X.: Parallel Implementation of the Accelerated Simulated Tempering Method. Proceedings of 3rd NPSC Conference, Atlanta (2006)
14. Li, Y., Protopopescu, V.A., Arnold, N., Zhang, X., Gorin, A.: Hybrid Parallel Tempering/Simulated Annealing Method. submitted to Physical Review E (2006)
15. Du, Z., Li, S., Li, S., Wu, M., Zhu, J.: Massively parallel simulated annealing embedded with downhill – a SPMD algorithm for cluster computing. In: Proceedings of 1st IEEE Computer Society International Workshop on Cluster Computing (1999)
16. Drugan, M.M., Thierens, D.: Evolutionary Markov Chain Monte Carlo. Technical Report UU-CS-2003-047, Utrecht university (2003)
17. Liu, J.S., Sabatti, C.: Simulated Sintering: Markov Chain Monte Carlo with Spaces of Varying Dimensions. In: Bayesian Statistics 6, pp. 389–413. Oxford University Press, Oxford (1999)
18. Cercueil, A., Francois, O.: Monte Carlo simulation and population-based optimization. In: Congress on Evolutionary Computation, pp. 191–198 (2001)
19. Laskey, K.B., Myers, J.W.: Population Markov Chain Monte Carlo. Machine Learning , 175–196 (2003)
20. Mahfoud, S.W., Goldberg, D.E.: Parallel Recombinative Simulated Annealing: a Genetic Algorithm. Parallel Computing, 1–28 (1995)