Contents lists available at ScienceDirect

Parallel Computing

journal homepage: www.elsevier.com/locate/parco

A decentralized parallel implementation for parallel tempering algorithm

Yaohang Li [a,*], Michael Mascagni [b], Andrey Gorin [c]

[a] Department of Computer Science, North Carolina A&T State University, 1601 East Market Street, Greensboro, NC 27411, USA
[b] Department of Computer Science, Florida State University, Tallahassee, FL 32306, USA
[c] Division of Computer Science and Mathematics, Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA

ARTICLE INFO

Article history:
Received 27 March 2008
Received in revised form 29 November 2008
Accepted 10 December 2008
Available online 31 December 2008

Keywords:
Markov Chain Monte Carlo
Parallel tempering
Parallel random numbers

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 decentralized parallel implementation of PT using the message passing interface (MPI) and the scalable parallel random number generators (SPRNG) library. By taking advantage of the characteristics of pseudorandom number generators, this implementation eliminates global synchronization and reduces the overhead caused by interprocessor communication in replica exchange in PT. Moreover, our proposed non-blocking replica exchange reduces communication overhead in pair-wise process replica exchanges by allowing the process reaching the replica exchange point to leap-ahead while waiting for the other one to reach the common replica exchange point. Also, temperature exchange instead of conformation replica exchange is proposed to reduce communication and achieve load balancing in the participating processors in the PT computation. All these enable one to efficiently apply PT to large-scale massively parallel systems. The efficiency of this parallel PT implementation is demonstrated in the context of minimizing various benchmark functions with complicated landscapes as objective functions. Our computational results and analysis have shown that the decentralized PT is scalable, reproducible, load-balanced, and yields insignificant communication overhead.

1. Introduction

Parallel tempering (PT) [40], also known as replica exchange or the Multi-Markov Chain method, is a powerful Markov Chain Monte Carlo (MCMC) sampling scheme proposed by Marinari and Parisi [1], and Geyer and Thompson [2]. In PT, 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 exploring 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, replicas at neighboring temperature levels are allowed to exchange configurations from time to time, subject to an acceptance criterion. By carefully setting up the set of temperatures used, and the number of replicas, PT can reduce the relaxation time of the Monte Carlo simulations in the physical systems, and can improve the convergence to a global minimum. Therefore, PT is more favorable than running multiple Markov Chain Monte Carlo simulations without replica exchanges. PT is ideal for complex physical systems that are characterized by rough energy landscapes. Successful PT applications include the simulation of biomolecules [3], the determination of X-ray structures [4], polymer simulation [5], and structure prediction in small proteins [6,7].

* Corresponding author. Tel.: +1 3363347245l; fax: +1 3362247244.
E-mail address: yaohang@ncat.edu (Y. Li).

0167-8191/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.parco.2008.12.009

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 or core [13]. 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 incurred when exchanging the replicas. The recent study shows that PT scales only to about 32 replicas in protein folding applications [37,38].

In this paper, we present a novel decentralized replica exchange PT implementation based on our previous work [36] with more generalized scheme and advanced overhead reduction techniques. Our implementation takes advantage of the MPI and scalable parallel random number generators (SPRNG) [8,35] libraries. Functions in the MPI library are used for the necessary interprocessor communication in the parallel computing environment; while 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 exchange decision without performing interprocessor communication. Non-blocking replica exchange is implemented to allow a process to perform leap-ahead local transitions while waiting for replica exchange communication with the other process. Moreover, temperature exchange instead of configuration exchange is used to reduce the amount of communication when replica exchange does occur, which leads to a more generalized random replica exchange scheme with temperature exchange. All these efforts lead to an efficient and scalable decentralized implementation of replica exchange in PT, where the interprocessor communication overhead due to replica exchange is minimized.

The rest of the paper is organized as follows. Sections 2 and 3 illustrate the general PT scheme and the decentralized implementation of PT, respectively. Section 4 presents the computational results of decentralized PT on various benchmark functions. Efficiency, scalability, reproducibility, replica exchange schemes, and load balancing of decentralized PT are analyzed in Section 5. Section 6 discusses the extension of decentralized PT to various extended PT algorithms. Section 7 summarizes our conclusions and future research directions.

2. The parallel tempering scheme

In 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. The 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

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

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 temperature level, a Markov chain is constructed to sample $E(.)$ 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 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

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

The new state is accepted with the probability $\min(1, w_{\text{Local}}(x_i \to x_i'))$. The detailed balance condition holds for each replica in the 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 $i+1$.

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

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

$$\begin{aligned} P_{\text{Replica}}(x_i \leftrightarrow x_{i+1}) &= P(\{x_1, \ldots, x_i, x_{i+1}, \ldots, x_N\} | \{x_1, \ldots, x_{i+1}, x_i, \ldots, x_N\}) \\ &= \min\left(1, \frac{\Pi(\{x_1, \ldots, x_{i+1}, x_i, \ldots, x_N\})}{\Pi(\{x_1, \ldots, x_i, x_{i+1}, \ldots, x_N\})}\right) \\ &= \min(1, e^{-\beta_i E(x_{i+1}) - \beta_{i+1} E(x_i) + \beta_{i+1} E(x_{i+1}) + \beta_i E(x_i)}). \end{aligned}$$

Generally, the relaxation time [9] is determined by the escape time from the minima in the objective function landscape. In effect, the replica exchange transition enables replicas to be cooled down and warmed up. This allows a replica at low temperature level to have a chance to reach a higher temperature level, where it will have a higher probability to escape from a local minimum. As a result, with an appropriate temperature ladder in parallel tempering, replica transitions can dramatically shorten the relaxation time.

Using the definition of the replica exchange probability and keeping θ as a constant, the detailed balance equation can be obtained for replica transition.

$$\begin{aligned} &P(\{x_1,\ldots,x_i,x_{i+1},\ldots,x_N\}|\{x_1,\ldots,x_{i+1},x_i,\ldots,x_N\})\Pi(\{x_1,\ldots,x_{i+1},x_i,\ldots,x_N\}) \\ &= P(\{x_1,\ldots,x_{i+1},x_i,\ldots,x_N\}|\{x_1,\ldots,x_i,x_{i+1},\ldots,x_N\})\Pi(\{x_1,\ldots,x_i,x_{i+1},\ldots,x_N\}) \end{aligned}$$

A descriptive pseudo-code listing of the PT algorithm is as follows:

```
Initialize $N$ replica $x_1$, $x_2$, ..., $x_N$ and their corresponding temperatures $T_1$, $T_2$, ..., $T_N$
Initialize $t \leftarrow 0$
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 $U_M \leqslant w_{local}(x_i \rightarrow x_i')$ then $x_i \leftarrow x_i'$

    }
    //Perform Replica Transition
    Sample a uniform [0, 1) random variable $U_R$
    if $U_R \leqslant \theta$ then {
        Sample an integer variable $i$ from $U[1,N-1]$
        Sample a uniform [0, 1) random variable $U_S$
        if $U_S \leqslant P_{\text{Replica}}(x_i \leftrightarrow x_{i+1})$ then
            $x_i \leftrightarrow x_{i+1}$
    }
    Increment $t$
} Until convergence is observed based on relaxation estimation
```

3. Decentralized parallel tempering 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 a replica transition should occur. The common decision is made using a random number uniformly distributed in the interval [0, 1], U[0, 1]. Instead of producing a U[0, 1] 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 same 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 the 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 some 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. By configuring the random number generators in the SPRNG library properly, independence of the parallel random number streams used in a parallel PT implementation can be ensured [8,12].

3.2. 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	Propose a new configuration x_i' 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

3.3. Efficient parallel PT implementation

A naïve implementation of PT is to adopt a master-slave paradigm, where a designated master process will determine which two slave processes will participate in the replica exchange operation, and then the two selected slave processes attempt replica exchange. The naïve implementation is centralized, which requires global synchronization and leads to significant interprocessor communication overhead. Moreover, the centralized implementation is clearly not scalable to large numbers of processes.

The goal of our decentralized replica exchange PT implementation is to eliminate the global synchronization for replica exchange operations. Fig. 1 shows a 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 the participant stream are globally shared, and so these random number sequences are exactly the same in all processes. If the random numbers chosen decide that a replica transition will occur, then random numbers are generated in the participant stream to determine which two processes will participate in replica exchange. The non-selected processes skip the replica transition. For the two randomly selected participant processes, the non-blocking replica exchange attempt, which will be described in Section 3.4, is then performed. 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 information required is that for the exchange of temperature and the energy function value.

3.4. Non-blocking replica exchange

By taking advantage of the reproducibility of pseudorandom numbers, global synchronization is eliminated and synchronization in replica exchange operations is limited to the two participant processes in a decentralized manner. To reduce the synchronization overhead more aggressively, we propose a non-blocking replica exchange scheme with leap-ahead moves.

In practical applications using PT, a replica exchange is usually attempted when the probability of exchange is rather small. Although the two participant processes carry out the same number of local Metropolis transitions before the replica exchange operation occurs, they probably do not actually reach the point of replica exchange at the same time due to various reasons, such as different acceptance rates, different numbers of replica exchange operations that occurred before, different objective function evaluation times, and performance difference of their processors. Therefore, the process that reaches the replica exchange point first may have to wait for the other process – possibly for a long period of time. Given that the proposed replica exchange has a chance, which is usually high, of not being accepted, in non-blocking replica exchange, the process which reaches the replica exchange point first does not wait for the slower one. Instead, it carries out the leap-ahead moves – the process sends out its conformation information used to perform replica exchange using the MPI function `MPI_Send()`, and performs a checkpoint operation by saving the status of its random number streams (proposal stream, local acceptance stream, and replica exchange stream) using the SPRNG function *pack_sprng()*, and other necessary variables, such as the iteration counter and current energy value $E(x)$. Afterward, it continues its local Metropolis transitions as if the replica exchange attempt was not accepted until the conformation information from the MPI non-blocking message-receiving function `MPI_IRecv()` or the next replica exchange transition occurs. Then, it tests the replica exchange condition. If the replica exchange condition is satisfied and replica exchange does occur, it abandons the move-ahead local Metropolis transitions and recovers the saved checkpoint conformation and random number streams information; otherwise, it continues with its local Metropolis transitions. Fig. 2 shows the flowchart of the non-blocking replica exchange scheme. The non-blocking replica exchange scheme allows the process that first reaches the replica exchange point to carry out its local Metropolis transitions ahead by assuming that the replica exchange attempt will not be accepted while waiting for the conformation information from the other process.

When the replica exchange attempt is accepted, the non-blocking replica exchange will lead to a rolling back overhead of recovering the saved conformation from checkpointing. The replica exchange acceptance rate is typically low – Rathore et al. [22] indicate that the optimal replica exchange acceptance rate is around 20%. If this optimal replica exchange acceptance

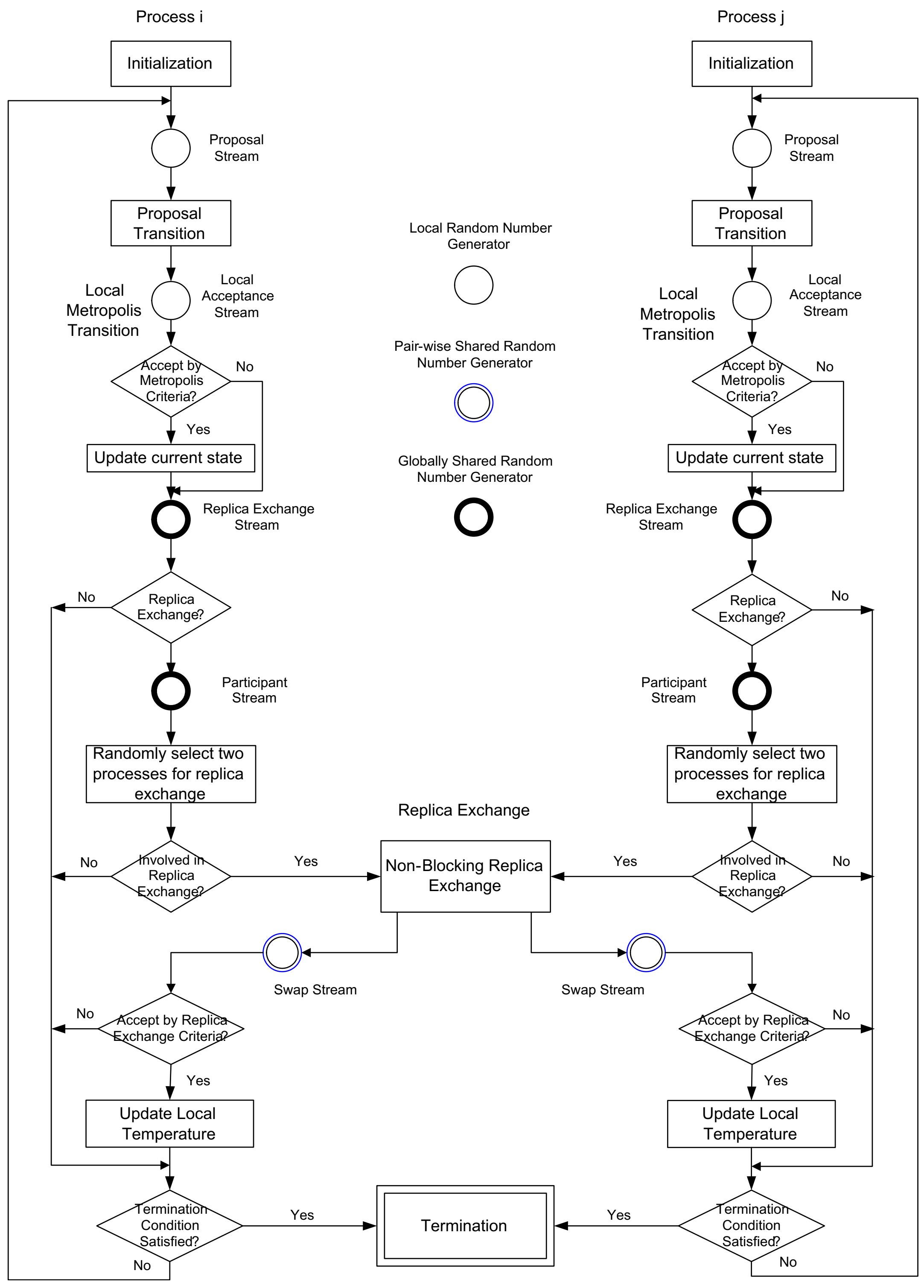

Process i
Process j
Initialization
Proposal Stream
Proposal Transition
Local Acceptance Stream
Local Metropolis Transition
Accept by Metropolis Criteria?
No
Yes
Update current state
Replica Exchange Stream
Replica Exchange?
Participant Stream
Randomly select two processes for replica exchange
Involved in Replica Exchange?
Replica Exchange
Non-Blocking Replica Exchange
Swap Stream
Accept by Replica Exchange Criteria?
Update Local Temperature
Termination Condition Satisfied?
Termination
Local Random Number Generator
Pair-wise Shared Random Number Generator
Globally Shared Random Number Generator

Fig. 1. Flowchart of the Decentralized replica exchange PT Scheme.

rate is adopted in the PT application, then 80% of the leap-ahead computations will contribute to the overall computation and only 20% will lead to rolling back operations. Moreover, in many PT applications, evaluating the system energy function via the objective function is rather costly. In comparison, the rolling back overhead is usually small and can be practically neglected.

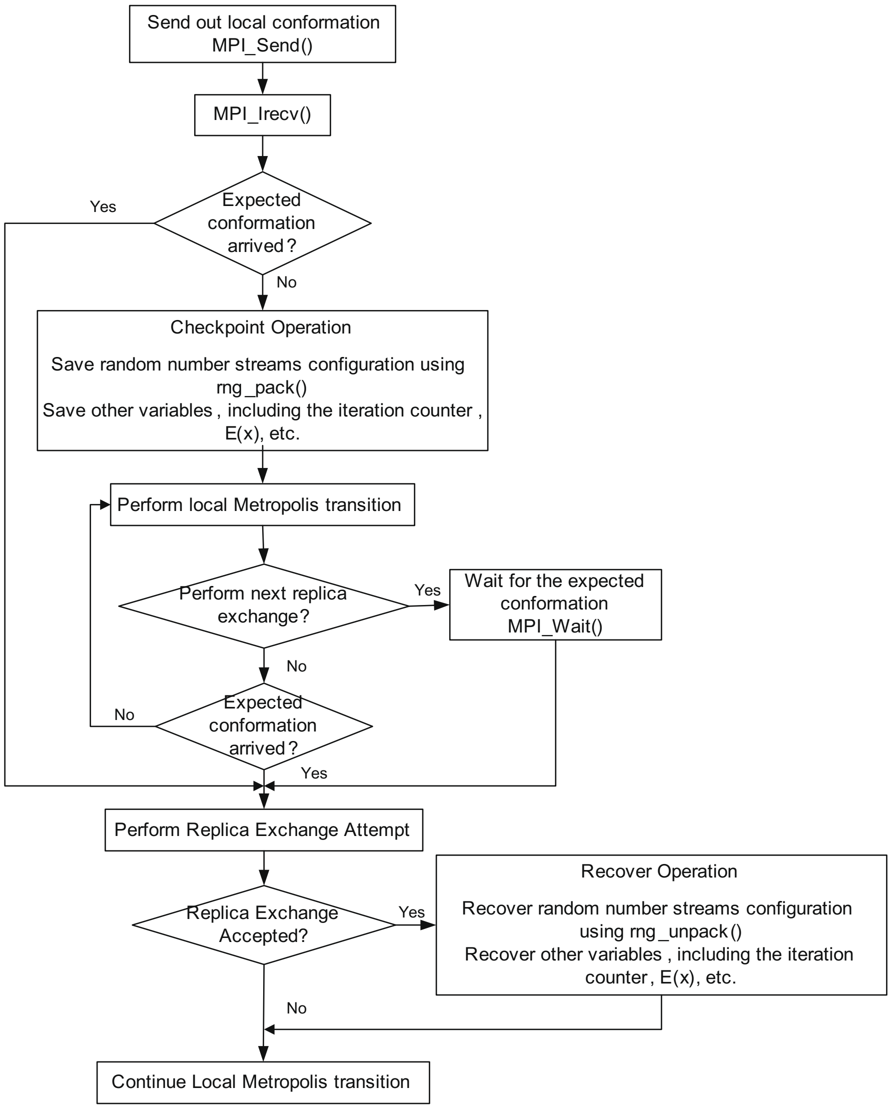

Send out local conformation
MPI_Send()
MPI_Irecv()
Expected conformation arrived?
Yes
No
Checkpoint Operation
Save random number streams configuration using rng_pack()
Save other variables, including the iteration counter, E(x), etc.
Perform local Metropolis transition
Perform next replica exchange?
Yes
Wait for the expected conformation MPI_Wait()
No
Expected conformation arrived?
No
Yes
Perform Replica Exchange Attempt
Replica Exchange Accepted?
Yes
Recover Operation
Recover random number streams configuration using rng_unpack()
Recover other variables, including the iteration counter, E(x), etc.
No
Continue Local Metropolis transition

Fig. 2. Flowchart of the Non-blocking replica exchange.

The leap-ahead computations will lead to wasted CPU cycles when rolling back operations occur. Assuming that the objective function evaluation times are normally distributed with mean, m, and standard deviation, σ, and the other operation times and interprocess communication times can be ignored, the maximum time deviation between two Markov chains before the replica exchange is $6\sigma\sqrt{1/\theta}$ in more than 99.9% of the situations, given a replica exchange probability of θ. With an acceptance rate of ρ_i, the maximum number of wasted leap-ahead objective function evaluations per replica exchange in a Markov chain, W_{waste}, at temperature level T_i, can be estimated as

$$W_{\text{waste}} = \frac{6\sigma\rho_i\sqrt{1/\theta}}{m}.$$

If we consider a scenario where the mean objective function evaluation time m is 100 units, σ is 10 units, θ is 0.001, and ρ_i is 25%, the maximum number of wasted leap-ahead objective function evaluations is approximately 5. Because the average number of objective function evaluations per replica exchange is normally around 1,000, thus, the wasted computational effort does not exceed 0.5% of the overall computational time in most cases (>99.9%).

Our wasted computational effort estimation is based on the assumption that the temperature will not have a significant effect on the objective function evaluation times, i.e., m is a constant. In some applications where the objective function evaluation times significantly differed at different temperature levels, such as the computation examples described in [41], our estimation is not applicable. In these applications, the CPU idle times may reach as high as 90% in one replica–one processor method [41] – using leap-ahead computation may lead to a large number of wasted CPU cycles. In these cases, allocating multiple replicas to a processor to increase CPU utilization [41] will be a more efficient solution than leap-ahead computation.

4. Computational results on benchmark objective functions

We carried out PT computations on a set of benchmark objective functions with different objective function landscapes. These benchmark functions include Ackley's function [30], Rosenbrock's function [33], Schwefel's function [32], Rastrigin's function [34], and Griewank's function [31], whose definitions and landscape characteristics are listed in Table 2. In our PT computations, a simple Gaussian proposal function is employed in the local Metropolis transition:

$$x_i' = x_i + g\Delta x,$$

where g is a standard normally distributed random variable with mean zero and variance one, and Δx is the transition step size. At the lowest temperature level, Δx is set to be $|x_{\max} - x_{\min}|/10^6$, while Δx is increased by a factor of 2 for every higher temperature level [10]. In local Metropolis transitions, we adopt an adaptive temperature scheme, where the temperature values are adjusted to satisfy the optimal acceptance rate ($20\% \sim 25\%$) in the Metropolis algorithm [39]. The interprocessor probability θ is set to be 0.01.

We compare parallel Metropolis, decentralized PT, and centralized PT in each benchmark function in 10-dimensions, 100-dimensions, and 1000-dimensions. Parallel Metropolis, decentralized PT, and centralized PT employ the same configuration of the temperature ladder and Metropolis transition functions. Centralized PT is implemented using the master-slave paradigm alluded to above, where replica exchange is achieved using MPI collective communication operations, which impose global synchronization overhead. Decentralized PT uses the approaches described in this paper. The only difference between parallel Metropolis and PT is that parallel Metropolis does not carry out the replica exchange operation, which is truly embarrassingly parallel, without any interprocessor communication overhead.

Table 3 shows the average job completion time of 10 runs (10^7 iterations for each run) of each algorithm on each benchmark function. The interprocessor communication overhead of replica exchange in PT is calculated against parallel Metropolis, which is embarrassingly parallel. In 10-dimensional benchmark functions, the function evaluation time is short, and the

Table 2
List of benchmark objective functions and their landscape characteristics.

Function	Definition	Landscape characteristics
Ackley	$f_{Ack}(x) = 20 + e - 20e^{-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}} - e^{\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)}$ $x_i \in [-30, 30], x^* = (0, 0, \ldots, 0), f_{Ack}(x^*) = 0$	A multi-modal function with many local minima disposed around the deep global minimum
Rosenbrock	$f_{Ros}(x) = \sum_{i=1}^{n-1}(100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2)$ $x_i \in [-2.048, 2.048], x^* = (1, 1, \ldots, 1), f_{Ros}(x^*) = 0$	A long narrow valley presented in the function landscape, where the global minimum is located at the bottom of the valley
Schwefel	$f_{Sch}(x) = 418.9829n + \sum_{i=1}^{n} x_i \sin(\sqrt{\lvert x_i\rvert})$ $x_i \in [-500, 500]$, $x^* = (-420.9687, \ldots, -420.9687), f_{Sch}(x^*) = 0$	The global minimum is geometrically distant, over the function landscape, from the next best local minima
Rastrigin	$f_{Ras}(x) = 10n + \sum_{i=1}^{n}(x_i^2 - 10\cos(2\pi x_i))$ $x_i \in [-5.12, 5.12], x^* = (0, 0, \ldots, 0), f_{Ras}(x^*) = 0$	Multi-modal function with widespread local minima regularly distributed in the function landscape
Griewank	$f_{Gri}(x) = 1 + \sum_{i=1}^{n}\frac{x_i^2}{4000} - \prod_{i=1}^{n}\cos(x_i/\sqrt{i})$ $x_i \in [-600, 600], x^* = (0, 0, \ldots, 0), f_{Ack}(x^*) = 0$	Multi-modal function with widespread local minima regularly distributed in the function landscape

Table 3
Average job completion time of parallel metropolis, decentralized PT, and centralized PT on 10, 100, and 1000 dimensional benchmark functions. Overheads of decentralized PT and centralized PT on parallel metropolis are also shown.†

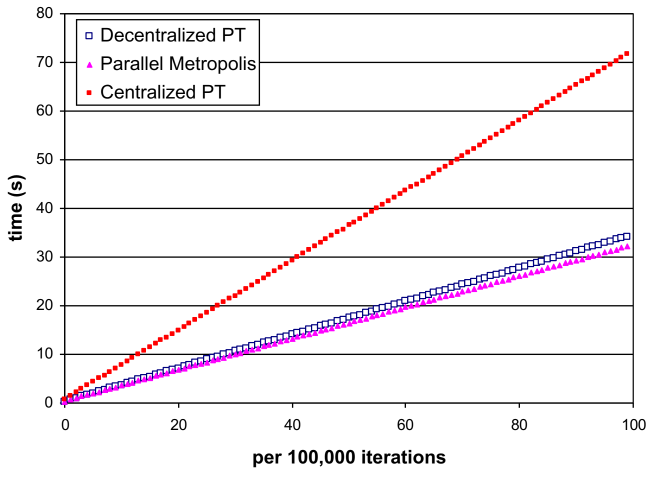
Function	Dimension	Average job completion time in seconds (overhead)		
		Parallel metropolis	Decentralized PT	Centralized PT
Ackley	10	32.19	34.26 (6.43%)	71.65 (122.58%)
	100	269.91	272.50 (1.06%)	386.71 (43.38%)
	1000	2603.91	2625.31 (0.82%)	3257.72 (25.11%)
Rosenbrock	10	21.60	23.53 (8.94%)	61.35 (184.03%)
	100	192.62	195.19 (1.33%)	322.73 (67.55%)
	1000	1903.44	1905.17 (0.09%)	2926.41 (53.74%)
Rastrigin	10	33.17	34.97 (5.26%)	68.90 (107.71%)
	100	300.06	301.84 (0.59%)	391.51 (30.48%)
	1000	3017.53	3022.51 (0.17%)	3779.89 (25.26%)
Schwefel	10	34.91	36.36 (4.15%)	73.25 (109.83%)
	100	321.08	325.69 (1.44%)	445.18 (38.65%)
	1000	3173.70	3184.38(0.34%)	4129.03 (30.10%)
Griewank	10	37.20	39.08 (5.05%)	75.63 (103.31%)
	100	340.55	345.26 (1.38%)	471.25 (38.38%)
	1000	3435.51	3438.36 (0.08%)	4422.68 (28.73%)

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

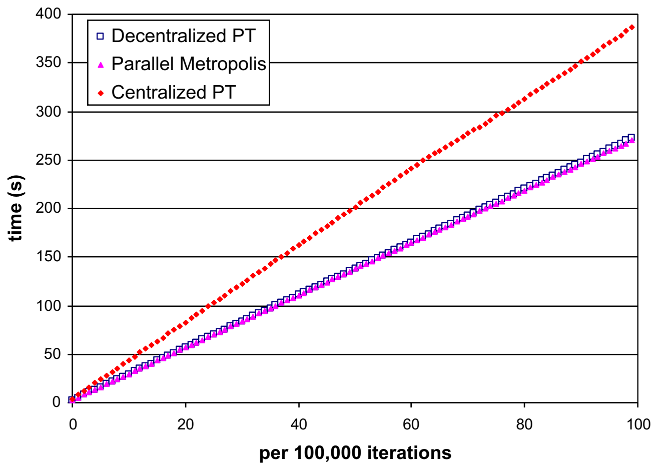
interprocessor communication time is rather significant. As a result, centralized PT yields a large synchronization overhead, which is over 100%, due to the global synchronization requirement in its replica exchange implementation. In decentralized PT, the communication overhead is less than 10% in all benchmark functions, since replica exchange operations are decentralized. In 100-dimensional functions, the interprocessor communication overhead is moderate compared to the function evaluation time. In centralized PT, the communication overhead is reduced, but it still remains in the range of 30–70% in all benchmark functions. In contrast, the overhead of decentralized PT is less than 2%. In 1000-dimensional functions, the function evaluation time is rather long, and the actual interprocessor communication overhead is relatively less significant. The synchronization overhead in centralized PT continues to be reduced, but still remains rather costly: over 20% in all benchmark functions. In decentralized PT, the interprocessor communication overhead is less than 1% in all benchmark functions, and average job completion time is almost indistinguishable from parallel Metropolis. Fig. 3a and b, and c show the performance comparison of centralized PT, decentralized PT, and parallel Metropolis on 10D, 100D, and 1000D Ackley's functions. One can find that compared to centralized PT, the interprocess communication overhead is significantly reduced in decentralized PT, showing a similar computational performance to the parallel Metropolis, which is embarrassingly parallel. Fig. 3b and c also show that in 100D and 1000D Ackley's functions where the function evaluation time is more significant than 10D function, the computational times of decentralized PT are almost overlapped on those of parallel Metropolis. Similar performance figures can be found in other benchmark functions.

It is well known that PT can reduce the relaxation time in Monte Carlo simulations and can accelerate the convergence to the ground state with an appropriately chosen temperature ladder [22], because the exchange between conformations in different replicas facilitates relaxation of conformations that might otherwise be trapped in local energy minima. Fig. 4 shows 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 using the 100-dimensional Rosenbrock function. 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 the global minimum compared to parallel Metropolis due to the reduction of the relaxation time afforded by the replication exchange transitions in PT. Similar improved convergence behavior of PT can also be found in computational experiments using other benchmark objective functions.

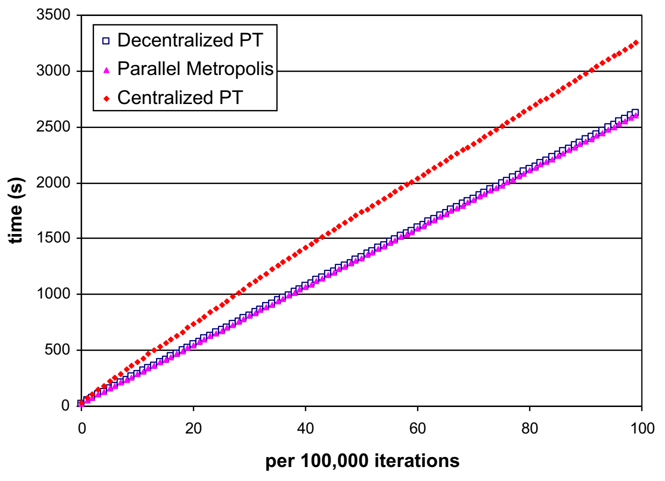
Table 4 shows the numbers and percentages of accepted function evaluations and wasted function evaluations in leap-ahead moves of the non-blocking replica exchange scheme using 10-dimensional, 100-dimensional, and 1000-dimensional Ackley functions, respectively. In non-blocking replica exchange, more function evaluations are carried out than the required number of function evaluations. The function evaluations in leap-ahead moves are carried out when one process is waiting for the other processes to reach to a common point for replica exchange operation. In this computational experiment, more than 5% of the function evaluations in leap-ahead moves are accepted and contributed to the overall PT computation; only a small fraction (less than 0.5%) of the function evaluations in leap-ahead moves is wasted. This agrees with our estimation of wasted computational effort provided in Section 3.4. It is important to notice that except for the usually insignificant overhead of recovering the checkpointed conformation and random number streams, the wasted function evaluations in leap-ahead moves of the non-blocking replica exchange scheme insignificantly delay the overall PT computation completion time in this computational example.



(a) 10D Ackley's Function



(b) 100D Ackley's Function



(c) 1000D Ackley's Function

Fig. 3. Performance Comparison of Decentralized Parallel PT, Parallel Metropolis, and Centralized PT in 10D, 100D, and 1000D Ackley's Functions on 8 processors.

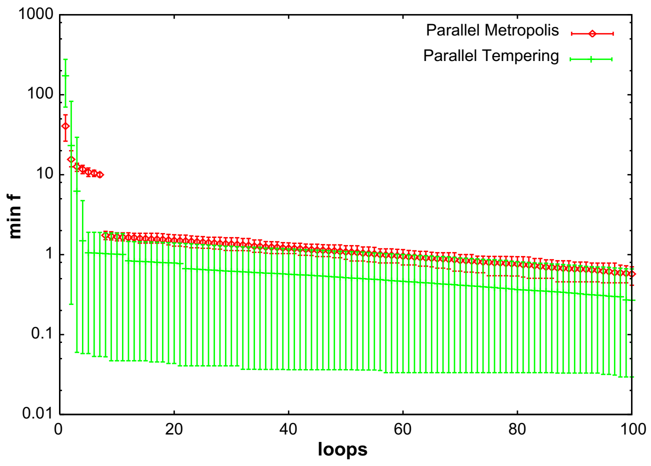


Fig. 4. Experimental Runs of Parallel Metropolis and parallel tempering on the 100-Dimensional Rosenbrock's Function. Each loop includes 10^5 iterations and shows the best, worst, and average function values.

Table 4
Number and percentages of accepted leap-ahead function evaluations and wasted leap-ahead function evaluations in non-blocking replica exchange in optimization computations of 10-, 100-, and 1000-dimensional Ackley functions.

	Actual function evaluations	Required function evaluations	Accepted function evaluations in leap-ahead moves		Wasted function evaluations in leap-ahead moves		Rollback (#)
			(#)	(%)	(#)	(%)	
10D	80,395,396	80,000,000	5,544,668	6.90	395,396	0.492	7,406
100D	80,363,105	80,000,000	4,709,746	5.86	363,105	0.451	4,478
1000D	80,350,262	80,000,000	4,180,208	5.20	350,262	0.436	4,074

5. Discussions

5.1. Correctness

Using various statistically independent but reproducible pseudorandom number sequences in decentralized PT preserves the order of relevant Metropolis transitions and replica transitions. In non-blocking replica exchange, the pseudorandom number status is saved by the SPRNG checkpoint function when work-ahead Metropolis transitions are attempted, and will be recovered to maintain the previous random number sequence if replica exchange attempt is accepted. As a result, the decentralized PT will yield exactly the same computational results as centralized PT or sequential PT, provided that the same random number sequences and initial random number seeds are used.

5.2. 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.

5.3. Reproducibility

Notice that the simulation of this parallel implementation of PT is reproducible. First, all SPRNG random number streams involved can be exactly reproduced by retrieving the same parameters and seeds in each pseudorandom number generator. Second, in each process, the Metropolis transitions can be reproduced by reproducing the local random numbers in the proposal stream and the local acceptance stream. Third, 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. Fourth, in non-blocking replica exchange, the random number streams are saved by the `rng_pack()` function in the SPRNG library, and can be recovered by the `rng_unpack()` function if replica exchange is accepted. Finally, when a replica exchange is attempted, each process pair can be reproduced by reproducing the corresponding swap stream.

5.4. Scalability

In our decentralized PT implementation, the replica exchange operations are decentralized and the interprocessor communication is minimized, which allows decentralized PT to run on scalable systems. Fig. 5 shows the computation time of a 100-dimensional Ackley function using decentralized PT, centralized PT, and parallel Metropolis on NCSA's Tungsten clusters with 2–64 processors. Parallel Metropolis is completely scalable because no interprocessor communication is involved. We notice that the computational time in centralized PT increases significantly as the number of processors increases, because global synchronization is more costly with large numbers of processors. In contrast, decentralized PT yields performance almost identical to that of parallel Metropolis as the number of processors increases.

5.5. Temperature exchange vs. replica conformation exchange

Replica exchange is employed in the PT scheme for improving mixing among the Markov chains running at various temperature levels. Replica exchange requires that the system pass configuration information between the 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 the large amount of interprocessor communication required to move the large amount of information required to specify the system configuration. 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 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.

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 the 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.

$$x_i \leftrightarrow x_j.$$

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

$$\begin{aligned}P_{R\ eplica}(x_i \leftrightarrow x_j) &= P(\{x_1,\ldots,x_i,\ldots,x_j,\ldots,x_N\}|\{x_1,\ldots,x_j,\ldots,x_i,\ldots,x_N\})\\ &= \min\left(1,\frac{\Pi(\{x_1,\ldots,x_j,\ldots,x_i,\ldots,x_N\})}{\Pi(\{x_1,\ldots,x_i,\ldots,x_j,\ldots,x_N\})}\right)\\ &= \min(1, e^{-\beta_i E(x_j)-\beta_j E(x_i)+\beta_j E(x_j)+\beta_i E(x_i)})\end{aligned}$$

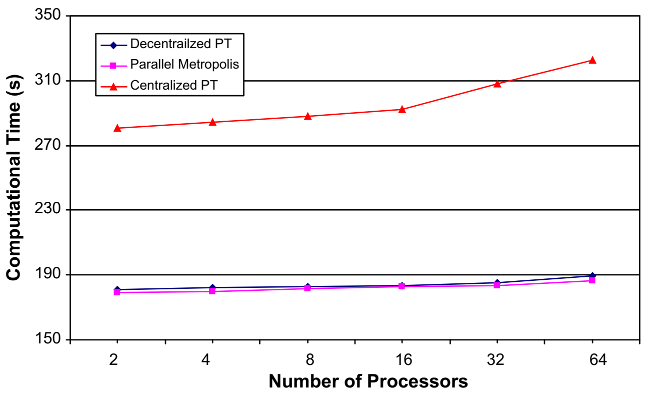


Fig. 5. Scalability of Decentralized PT, Centralized PT, and Parallel Metropolis on the NCSA Tungsten Clusters. The significant computational time increase in parallel Metropolis is due to the memory bus bandwidth limit problem in NCSA Tungsten clusters.

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

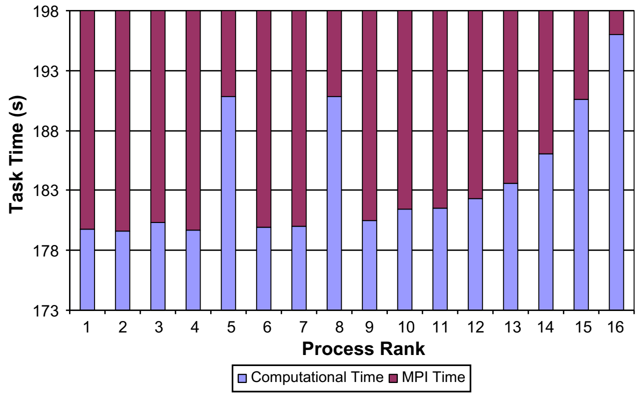
$$P(\{x_1,\ldots,x_i,\ldots,x_j,\ldots,x_N\}|\{x_1,\ldots,x_j,\ldots,x_i,\ldots,x_N\})\Pi(\{x_1,\ldots,x_j,\ldots,x_i,\ldots,x_N\})$$
$$= P(\{x_1,\ldots,x_j,\ldots,x_i,\ldots,x_N\}|\{x_1,\ldots,x_i,\ldots,x_j,\ldots,x_N\})\Pi(\{x_1,\ldots,x_i,\ldots,x_j,\ldots,x_N\})$$

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].

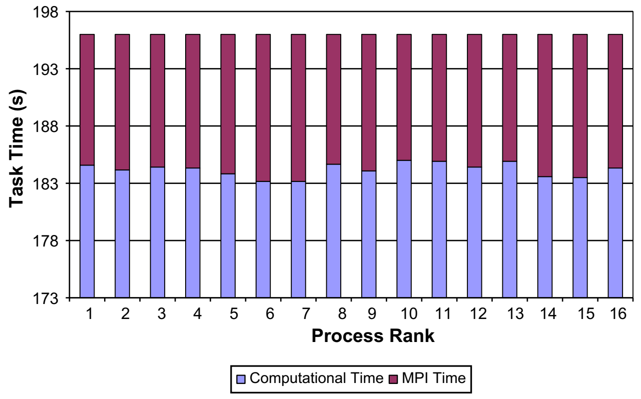
It is important to notice that temperature exchange (random replica exchange) may not be suitable for some applications because it may lead to direct replica exchange between low temperature and very high temperature. In some applications, the energy function landscape at high temperature is drastically different from that at low temperature. Direct replica exchange between low temperature and very high temperature may lead to the point that some conformational states involved in the equilibrium are no longer statistically significant [42]. In these applications, sufficient temperature levels between the high and low temperatures and replica exchanges at neighboring temperatures are required. However, if an application is suitable for temperature exchange, using temperature exchange will lead to further parallel efficiency improvement.

5.6. Load balancing

Another advantage of temperature exchange is to achieve load balancing in a parallel PT implementation. In PT, the computational load in each processor is different due to different acceptance rates, initial conformations, and random trajectories; however, typically, the processor carrying out Metropolis transitions at high temperature has a heavier computational load because of a higher acceptance rate, i.e., a more frequent update of its conformation. Temperature exchange allows the transition



(a) Decentralized PT with Conformation Replica Exchange



(b) Decentralized PT with Temperature Exchange

Fig. 6. Computational Time and MPI Time in Each Processor characterized using MPIp library. The computations were carried out on the NCSA Tungsten Clusters.

in high temperature to be distributed to a different processor. We use MPIp [21], a parallel programming performance tuning software library, to further characterize and compare decentralized PT with conformation replica exchange and temperature exchange. Fig. 6a and b show how the computation time and MPI time contributed to the overall job completion time in each processor of decentralized PT using conformation replica exchange and temperature exchange, respectively, in sampling a 100-dimensional Ackley function. In conformation replica exchange as shown in Fig. 6a, the computational time of each processor is unbalanced. Although there are two spikes in the middle, which is caused by the high acceptance rate probably due to other reasons such as initial locations, surrounding objective function landscape, or frequency of participation in replica exchange, the trend is that the computational time increases as the process rank increases, where high rank processes carry out simulations in high temperature. The most significant difference is 14 s. In contrast, in Fig. 6b, temperature exchange leads to a balanced computational load in each processor, where the most significant difference is less than 1 s.

6. Applications to extended PT algorithms

6.1. Hybrid PT/SA algorithm

The hybrid parallel tempering (PT)/simulated annealing (SA) algorithm [14] is a hybrid scheme combining PT and SA to achieve a fast barrier crossing capability, and gives one a higher probability of discovering the global minimum. In hybrid PT/SA, temperature at each level in the composite system is raised to a high value, and is then reduced gradually in the sampling process. The decentralized replica exchange scheme can be smoothly extended to the hybrid PT/SA algorithm with the addition of saving the temperature in the checkpoint operation and recovering it if the replica exchange attempt is not accepted.

6.2. Adaptable MCMC in PT scheme

In the PT scheme using adaptable MCMC, the temperature at each temperature level is no longer a constant. Instead, the temperature is adjusted to maintain a favorable acceptance rate ($20\% \sim 25\%$) [24] of the corresponding Markov chain. Decentralized replica exchange can be applied to the PT scheme with adaptable MCMC by saving the temperature value and acceptance rate in the checkpoint operation. To take advantage of temperature exchange, the acceptance rates need to be exchanged together with the temperature values.

6.3. Multicanonical replica exchange algorithm

The generalized ensemble algorithm [23] is a multicanonical ensemble method, whose fundamental idea is to "deform" the energy function to improve the rate of escape from the deep local minima. Herein, PT can be regarded as a special case of the parallel generalized ensemble, where the temperature variable is used to manipulate the energy function landscape. The multicanonical replica exchange algorithm [28] is an extension of parallel tempering algorithm, where the Metropolis transitions (canonical transition) in the local Monte Carlo move are replaced by multicanonical transitions. Correspondingly, a multicanonical potential energy E^{i}_{mu} is used at different temperature levels T_i instead of the same energy function E as in normal PT. Decentralized replica exchange can be extended to multicanonical replica exchange without any changes. To utilize temperature exchange, the parameters for the potential energy E^{i}_{mu} and E^{j}_{mu} for the two processes i and j participating in a replica exchange operation must be exchanged as well.

6.4. Replica exchange molecular dynamic

Replica exchange is also popularly used in molecular dynamics (MD) simulation [25–27], where the Metropolis–Hastings transitions at each temperature level are replaced by MD computations. The decentralized replica exchange scheme can be easily extended to replica exchange MD, where the proposal streams and local acceptance streams are no longer needed because the MD computation is deterministic. However, depending on the application requirements, independent local streams may be needed for configuration initialization.

7. Summary and future research directions

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 implementing the non-blocking replica exchange, we are able to eliminate the need for global synchronization and to minimize the interprocessor communication overhead. Our computational experiments, based on applying the decentralized PT implementation to various benchmark objective functions in low or high dimensions, including the Ackley, Rosenbrock, Rastrigin, Schwefel, and Griewank functions, indicate that an insignificant amount of interprocessor communication overhead contributed to the overall computational time. In conclusion, the decentralized PT is an efficient parallel implementation of the PT algorithm with minimized and insignificant interprocess communication overhead and balanced workload, which is scalable to a large number of processors. The decentralized PT method can also be easily applied to various extended PT algorithms, including hybrid PT/SA, adaptable MCMC in the PT scheme, multicanonical replica exchange, and replica exchange molecular dynamics.

In our future work, we plan to apply and study the performance of our decentralized PT approaches to various real-life applications, such as protein simulation, spin glasses [29], and similar problems. Moreover, the PT simulations we described in this paper are performed on parallel systems with distributed memory, and we plan to study the algorithm in shared memory systems in the future. The decentralized PT algorithm is possible to be implemented in shared memory systems, where similar reductions in synchronization overhead are likely to be achieved while message passing can be trivially implemented. Furthermore, study [41] shows that one replica per processor is not efficient. Combining optimal allocation of replicas with the decentralized PT may lead to further PT performance improvement.

Acknowledgements

Special thanks to Rick Kufrin at NCSA for helpful discussion in performance tuning using MPIp. The work is partially supported by NSF under Grant number CCF-0829382 to Y. Li, NCSA 2007 Summer Faculty Fellowship to Y. Li, 2008 ORAU/ORNL Summer Faculty Participation Program to Y. Li, and LDRD Program of the Oak Ridge National Laboratory managed by UT-Battelle, LLC, under Contract DE-AC05-00OR22725, to A. Gorin.

References

[1] E. Marinari, G. Parisi, Simulated tempering: a new Monte Carlo scheme, Europhysics Letters 19 (1992) 451–458.
[2] C.J. Geyer, E.A. Thompson, Annealing Markov Chain Monte Carlo with applications to ancestral inference, Journal of the American Statistical Association 90 (1995) 909–920.
[3] M. Falcioni, M.W. Deem, A biased Monte Carlo Scheme for zeolite structure solution, Journal of Chemical Physics 110 (1999) 1754–1766.
[4] A. Schug, T. Herges, A. Verma, W. Wenzel, Investigation of the parallel tempering method for protein folding, Journal of Physics: Condensed Matter 17 (2005) 1641–1650.
[5] A. Sikorski, Properties of star-branched polymer chains – Application of the replica exchange Monte Carlo method, Macromolecules 35 (18) (2002) 7132–7137.
[6] A. Schug, W. Wenzel, Predictive in-silico all atom folding of a four helix protein with a free energy model, Journal of the American Chemical Society 126 (2004) 16737.
[7] Y. Li, C.E.M. Strauss, A. Gorin, Parallel tempering in rosetta practice, in: Proceedings of International Conference on Bioinformatics and its Applications (ICBA'04), Fort Lauderdale, Florida, 2004.
[8] M. Mascagni, A. Srinivasan, Algorithm 806: SPRNG: A scalable library for pseudorandom number generation, ACM Transactions on Mathematical Software 26 (2000) 436–461.
[9] A. Sokal, Monte Carlo methods in statistical mechanics: foundations and new algorithms, in: Functional Integration (Cargèse, 1996), Plenum, New York, 1997, pp. 131–192.
[10] Y. Li, V.A. Protopopescu, A. Gorin, Accelerated simulated tempering, Physics Letters A 328 (4) (2004) 274–283.
[11] J.S. Liu, F. Liang, W.H. Wong, The Use of Multiple-Try Method and Local Optimization in Metropolis Sampling, Technical Report, Department of Statistics, Stanford University, 1998.
[12] A. Srinivasan, M. Mascagni, D. Ceperley, Testing parallel random number generators, Parallel Computing 29 (2003) 69–94.
[13] Y. Li, J. Clark, X. Zhang, Parallel implementation of the accelerated simulated tempering method, in: Proceedings of 3rd NPSC Conference, Atlanta, 2006.
[14] Y. Li, V.A. Protopopescu, N. Arnold, X. Zhang, A. Gorin, Hybrid Parallel Tempering/Simulated Annealing Method, submitted to Physical Review E, 2006.
[21] J.S. Vetter, M.O. McCracken, Statistical scalability analysis of communication operations in distributed applications, in: Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPOPP), 2001.
[22] N. Rathore, M. Chopra, J.J. de Pablo, Optimal allocation of replicas in parallel tempering simulations, Journal of Chemical Physics 122 (2005) 024111.
[23] U.H.E. Hansmann, Y. Okamoto, Generalized-ensemble Monte Carlo method for systems with rough energy landscape, Physical Review E 56 (20) (1997) 2228–2233.
[24] A. Kone, D.A. Kofke, Selection of temperature intervals for parallel temerping simulations, Journal of Chemical Physics 122 (20) (2005) 206101.
[25] Y. Sugita, Y. Okamoto, Replica-exchange molecular dynamics method for protein folding, Chemical Physics Letters 314 (1999) 141–151.
[26] W. Zhang, C. Wu, Y. Duan, Convergence of replica exchange molecular dynamics, Journal of Chemical Physics 123 (2005) 154105.
[27] M. Eleftheriou, A. Rayshubski, J.W. Pitera, B.G. Fitch, R. Zhou, R.S. Germain, Parallel implementation of the replica exchange molecular dynamics algorithm on blue Gene/L, in: Proceedings of Parallel and Distributed Processing Symposium, 2006.
[28] Y. Sugita, Y. Okamoto, Replica-exchange multicanonical algorithm and multicanonical replica exchange method for simulating systems with rough energy landscape, Chemical Physics Letters 329 (3) (2000) 261–270.
[29] H.G. Katzgraber, M. Palassini, A.P. Young, Monte Carlo simulations of spin glasses at low temperatures, Physical Review B 63 (2001) 184422.
[30] D. Ackley, An empirical study of bit vector function optimization, Genetic Algorithms and Simulated Annealing (1987) 170–215.
[31] A. Torn, A. Zilinskas, Global Optimization, Lecture Notes in Computer Science, vol. 350, Springer-Verlag, Berlin, 1989.
[32] H.P. Schwefel, Numerical Optimization of Computer Models, John Wiley & Sons, 1981.
[33] H.H. Rosenbrock, An automatic method for finding the greatest or least value of a function, Computer Journal 3 (1960) 175–184.
[34] L.A. Rastrigin, Extremal Control Systems, Theoretical Foundations of Engineering Cybernetics Series, vol. 3, Nauka, Moscow, 1974.
[35] M. Mascagni, J. Ren, New development in the scalable parallel random number generator (SPRNG) library, The Institute of Statistical Mathematics Cooperative Research Report 210 (2008) 120–125.
[36] Y. Li, M. Mascagni, A. Gorin, Decentralized replica exchange parallel tempering: an efficient implementation of parallel tempering using MPI and SPRNG, in: Proceedings of International Conference on Computational Science and Its Applications (ICCSA), Kuala Lumpur, 2007.
[37] A. Schug, T. Herges, W. Wenzel, Allatom folding of the three helix HIV accessory protein with an adaptive parallel tempering method, Proteins 57 (2004) 792.
[38] A. Schug, T. Herges, A. Verma, W. Wenzel, Investigation of the parallel tempering Method for Protein Folding 17 (2005) 1641–1650.

[39] G.O. Roberts, A. Gelman, W.R. Gilks, Weak convergence and optimal scaling of random walk metropolis algorithms, Annals of Applied Probability 7 (1997) 110–120.
[40] D.J. Earl, M.W. Deem, Parallel tempering: theory, applications, and new perspectives, Physical Chemistry Chemical Physics 7 (2005) 3910–3916.
[41] D.J. Earl, M.W. Deem, Optimal allocation of replicas to processors in parallel tempering simulations, Journal of Physical Chemistry B 108 (2004) 6844–6849.
[42] D.M. Zuckerman, E. Lyman, A second look at canonical sampling of biomolecules using replica exchange simulation, Journal of Chemical Theory and Computation 2 (2006) 1200–1202.