Optimal Allocation of Replicas to Processors in Parallel Tempering Simulations[†]

David J. Earl and Michael W. Deem*

Department of Bioengineering and Department of Physics & Astronomy, Rice University, 6100 Main Street—MS 142, Houston, Texas 77005-1892

Received: December 1, 2003; In Final Form: January 14, 2004

The optimal allocation of replicas to a homogeneous or heterogeneous set of processors is derived for parallel tempering simulations on multiprocessor machines. In the general case, it is possible without substantially increasing wall clock time to achieve nearly perfect utilization of CPU time. Random fluctuations in the execution time of each replica do not significantly degrade the performance of the scheduler.

1. Introduction

The parallel tempering, or replica exchange, Monte Carlo method is an effective molecular simulation technique for the study of complex systems at low temperatures.[1−3] Parallel tempering achieves good sampling by allowing systems to escape from low free energy minima by exchanging configurations with systems at higher temperatures, which are free to sample representative volumes of phase space. The use of parallel tempering is now widespread in the scientific community.

The idea behind the parallel tempering technique is to sample n replica systems, each in the canonical ensemble, and each at a different temperature, T_i. Generally $T_1 < T_2 < ... < T_n$, where T_1 is the low-temperature system, of which we are interested in calculating the properties. Swaps, or exchanges, of the configurational variables between systems i and j are accepted with the probability

$$p = \min\{1, \exp[-(\beta_i - \beta_j)(H_j - H_i)]\} \tag{1}$$

where $\beta_i = 1/(k_B T_i)$ is the reciprocal temperature, and H_i is the Hamiltonian of the configuration in system i. Swaps are typically attempted between systems with adjacent temperatures, $j = i + 1$. Parallel tempering is an exact method in statistical mechanics, in that it satisfies the detailed balance or balance condition,[4] depending on the implementation.

Because of the need to satisfy the balance condition, the n different systems must be synchronized whenever a swap is attempted. This synchronization is in Monte Carlo steps, rather than in real, wall clock time. In other words, all processors must finish one Monte Carlo step before any of the processors may start the next Monte Carlo step. In parallel tempering, a convenient definition of Monte Carlo step is the ordered set of all of the Monte Carlo moves that occur between each attempted swap move. These Monte Carlo moves are all of the individual moves that equilibrate each system in the parallel tempering ensemble, such as Metropolis moves, configurational bias moves, volume change moves, hybrid Monte Carlo moves, and so on. Rephrasing, the balance condition requires that at the beginning of each Monte Carlo step, each replica must have completed the same number of Monte Carlo steps and must be available to swap configurations with the other replicas. This constraint introduces a potentially large inefficiency in the simulation, as different replicas are likely to require different amounts of computational processing time in order to complete a Monte Carlo step. This inefficiency is not a problem on a single processor system, as a single processor will simply step through all the replicas to complete the Monte Carlo steps of each. This inefficiency is a significant problem on multiprocessor machines, however, where individual CPUs can spend large quantities of time idling as they wait for other CPUs to complete the Monte Carlo steps of other replicas.

Traditionally, each processor on multiprocessor machines has been assigned one replica in parallel tempering simulations. It is the experience of the authors that this type of assignment is generally highly inefficient, with typical CPU idle times of 40−60%. When one takes into account that the lower-temperature systems should have more moves per Monte Carlo step due to the increased correlation times, the idle time rises to intolerable levels that can approach 95%. The issue of idle time has not been previously addressed, and it is clear that a scheme which could allocate replicas to processors in an optimal manner would be useful.

In this paper we address the optimal allocation of replicas to CPUs in parallel tempering simulations. The manuscript is organized as follows. In section 2 we present the theory for the allocation of replicas to a homogeneous set of processors. In section 3 we present results where the theory is applied to several model examples. In section 4 we discuss our results, compare them with the conventional parallel tempering scheme, and consider the effects of including communication times and randomness in execution time into our analysis. We draw our conclusions in section 5. An appendix presents the theory for a heterogeneous set of processors.

2. Theory of Replica Allocation to Processors

In a parallel tempering simulation, balance requires that each replica system be synchronized at the start of each Monte Carlo step. Considering replica i, in every Monte Carlo step we will attempt $N_{\text{move}}(T_i)$ random Monte Carlo configurational moves, and the average real wall clock time to complete one Monte Carlo move is given by $\alpha(T_i)$. The total wall clock time for replica i to complete its Monte Carlo step is

$$\tau_i = \alpha(T_i) N_{\text{move}}(T_i) \tag{2}$$

As we have already stated, the simple allocation of one replica

* Corresponding author. E-mail: mwdeem@rice.edu. Fax: 713-348-5811.
[†] Part of the special issue "Hans C. Andersen Festschrift".

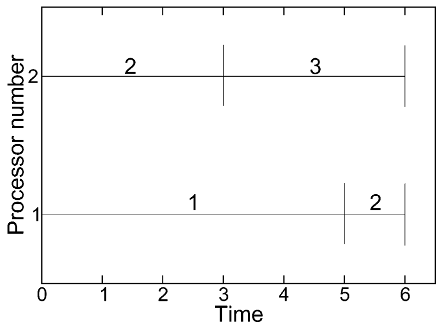

Figure 1. Simple example of the allocation of three replicas to two processors. In this example, an efficient allocation requires that replica 2 be split between processors 1 and 2. The replica numbers are marked on the figure.

to one processor for the entire simulation is inefficient. This is because α, the time per configurational move, depends on the temperature of the system. The value of α can typically vary by a factor of 3 or more between the fastest and the slowest system resulting in long idle times for the CPUs that are assigned to the higher temperature systems. The value of α varies because the composition of the configurational moves and their acceptance ratio varies with temperature. Typically, but not always, the highest temperature moves take less wall clock time on average to complete. Additionally, it is often desirable to perform more configurational Monte Carlo moves per Monte Carlo step at lower temperatures because the correlation time is longer than it is at higher temperatures. This makes the inefficiency of allocating one replica to one processor dramatically worse. In eq 2, $N_{\rm move}$ is a function of T_i to allow for the larger number of configurational moves that may be performed at lower temperatures. In most simulations that are currently performed, $N_{\rm move}$ is the same for all replicas because of the disastrous inefficiency implications of increasing $N_{\rm move}$ for low-temperature replicas, for which α is also often larger. Using an optimal allocation of replicas, the possibility of varying $N_{\rm move}$ for different replicas exists, as discussed in section 3 below.

The optimal allocation of replicas to processors is a nontrivial problem even in remarkably simple situations. For example, consider the case where $n = 3$, $\tau_1 = 5$, $\tau_2 = 4$, and $\tau_3 = 3$. Using three processors is clearly inefficient, as two processors would be idle while they are waiting for replica 3 to complete. The optimal allocation is to split one of the replicas on two processors, as shown in Figure 1. Only two processors are required, and they will both run at 100% efficiency if the replica is split correctly. Note that the splitting must be causally ordered in time. In the example of Figure 1, replica 2 is started on processor 2 and completed on processor 1 two time units after being stopped on processor 2.

A general replica scheduler can be derived starting with the assumptions that one replica cannot be simultaneously run on more than one processor and that one processor can only run one replica at a time, this second assumption being the simplest and, as it turns out, the most efficient use of the processors. The logic of the derivation comes from scheduling theory,[5,6] which is frequently used to solve problems of this type in operations research and industrial engineering. Given n replicas, where the time to complete replica i is τ_i, the total processing time required to complete all of the replicas is

$$W = \sum_{i=1}^{n} \tau_i \tag{3}$$

We let $\tau_{\rm long}$ be the CPU time of the longest replica. If we have

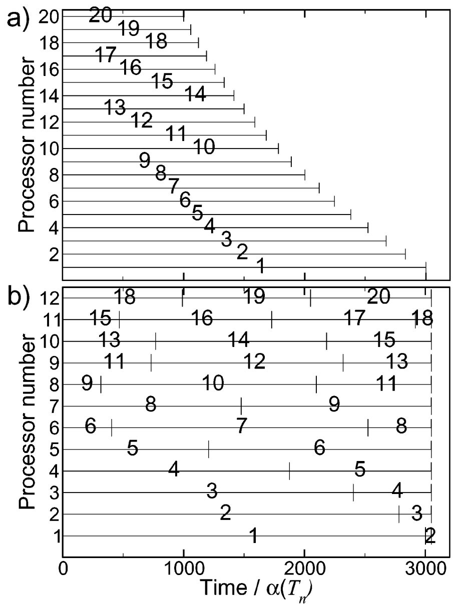

Figure 2. (a) Replica allocation in the traditional one replica per processor parallel tempering simulation using 20 replicas. (b) Assignment of the same replicas to processors as optimized by the scheduler derived in section 2. The replica numbers are marked on the figure.

X processors, then the shortest possible total wall clock time required to complete execution of all of the replicas is given by

$$\tau_{\rm wall} = \max(W/X, \tau_{\rm long}) \tag{4}$$

The optimum integer number of processors to achieve 100% theoretical efficiency will be

$$X^{(\rm N)} = \lfloor W/\tau_{\rm long} \rfloor \tag{5}$$

where $\lfloor y \rfloor$ is the largest integer equal to or less than the real number y. The number of processors required to achieve the minimum wall clock time will be

$$X^{(N+1)} = \lceil W/\tau_{\rm long} \rceil \tag{6}$$

where $\lceil y \rceil$ is the smallest integer equal to or greater than the real number y. The optimal allocation can either be done for minimum, zero percent, idle time, $X^{(N)}$, or minimum wall clock time, $X^{(N+1)}$. Having made the choice of one of these two numbers of processors, the optimal scheduler then proceeds by assigning the replicas sequentially to the first processor until that processor has filled its allocation of $\tau_{\rm wall}$ wall clock time. Typically this will result in the last replica allocated to the first processor being split, with the "remaining" time carried over to the second processor. The remaining replicas are sequentially allocated to the second processor, with again a possible split in the last replica allocated. This procedure is repeated until all the replicas have been allocated. In the minimum wall clock, $X^{(N+1)}$, case, the final processor will not be completely filled unless $W/X^{(N+1)} = \tau_{\rm long}$, and there will be a small amount of idle time. In the minimum idle time case, there will be no idle time. An example of how the scheduler assigns replicas to processors is shown in Figure 2 for a 20 replica case where $\tau_{\rm long}/\tau_{\rm short} = 3$, where $\tau_{\rm short}$ is the wall clock time of the replica that completes its Monte Carlo step most quickly.

It is immediately apparent that the scheduler[7] is extremely simple and very effective. The scheduler may easily be applied to existing parallel simulation codes. To apply the theory to a

TABLE 1: Results for the Parallel Tempering Job Allocation Optimized by the Scheduler for Run Time or Number of CPUs and for the Traditional Allocation

	X	I (%)	C (%)	I (%) $\gamma = 0.1$	C (%) $\gamma = 0.1$	I (%) $\gamma = 0.5$	C (%) $\gamma = 0.5$	I (%) $\gamma = 1.0$	C (%) $\gamma = 1.0$
				Example 1					
maximum efficiency	12	0.0	101.66	10.55 ± 0.02	113.81 ± 0.05	36.96 ± 0.06	163.26 ± 0.23	54.55 ± 0.10	227.59 ± 0.46
minimum run time	13	6.16	100.0	16.16 ± 0.02	112.10 ± 0.05	41.10 ± 0.06	161.47 ± 0.23	57.65 ± 0.10	225.71 ± 0.48
traditional	20	39.0	100.0	41.12 ± 0.03	104.12 ± 0.06	57.35 ± 0.06	147.18 ± 0.29	69.65 ± 0.07	208.72 ± 0.60
				Example 2					
maximum efficiency	3	0.0	128.20	4.65 ± 0.02	134.62 ± 0.07	19.29 ± 0.09	160.33 ± 0.33	34.50 ± 0.73	193.76 ± 0.66
minimum run time	4	3.93	100.0	9.81 ± 0.02	106.75 ± 0.06	27.49 ± 0.09	134.82 ± 0.28	43.34 ± 0.15	171.60 ± 0.55
traditional	20	80.77	100.0	80.62 ± 0.01	100.00 ± 0.10	82.55 ± 0.02	116.60 ± 0.31	86.41 ± 0.03	149.97 ± 0.48
				Example 3					
maximum efficiency	6	0.0	110.53	7.25 ± 0.02	119.28 ± 0.05	27.70 ± 0.07	154.55 ± 0.22	44.20 ± 0.13	200.60 ± 0.44
minimum run time	7	5.26	100.0	12.93 ± 0.03	108.95 ± 0.05	33.70 ± 0.10	144.92 ± 0.23	49.59 ± 0.15	191.20 ± 0.444
traditional	50	86.74	100.0	86.75 ± 0.01	100.78 ± 0.11	89.07 ± 0.03	126.69 ± 0.38	97.76 ± 0.03	169.56 ± 0.66

[a] Results are shown for the three example systems described in section 3. Shown are the number of processors (X), the percentage CPU idle time (I), and the wall clock time of the simulation relative to the results for the traditional allocation without randomness (C). Idle time and wall clock time are also shown for the case where the CPU time required for each replica is a stochastic quantity, eq 8, with $\gamma = 0.1$, 0.5, and 1.0.

practical simulation, one must first perform a short preliminary simulation for each replica to obtain an estimate of $\alpha(T_i)$, and hence τ_i from eq 2. We note that the scheduler could be run after each Monte Carlo step, since the calculation time associated with the scheduler is minimal. Such use of the scheduler would automatically lead to an adaptive or cumulative estimate of α. Note that at all times, the balance properties of the underlying Monte Carlo scheme are unaffected by the replica allocations of the scheduler. It is also worthy of comment that the scheduler could be run with parallel tempering in multiple dimensions, for example differing chemical potentials[8,9,10] or pair potentials[11] for each replica, in addition to variations in temperature. Increasing the number of order parameters that we use in the parallel tempering not only may improve sampling but also may provide a better estimate of α, since the estimate of α as a local function of phase space increases as the number of order parameters increases.

In this section we have derived the scheduler for a homogeneous cluster of processors. In the Appendix we derive a similar scheme for a heterogeneous cluster.

3. Results

In this section, we apply the optimal replica scheduler to three different parallel tempering simulation examples. Details of the three different examples are given below, and the performance of the scheduler can be seen in Table 1. Results are shown in the table for the minimum idle time, minimum wall clock time, and traditional one-replica-per-processor cases. For each case we show the number of processors used, the CPU idle time as a percentage of the overall time for one Monte Carlo step, and the real wall clock time for the simulation relative to that of the traditional parallel tempering approach. To motivate the parameter values chosen for the examples, we note that, in our experience with simulations of the 20−50 amino acid peptides from the innate immune system that are known as cystine-knot peptides, we find the ratio of correlation times between the low and high-temperature replicas can vary by a factor of 10^2−10^5, $N_{\text{move}}(T_1)/N_{\text{move}}(T_n) = 10^2$−$10^5$, on the order of $N_{\text{move}}(T_n) = 10^3$−$10^5$ configurational Monte Carlo moves are typically performed during each Monte Carlo step at the highest temperature, and $N_{\text{move}} = 10^6$ configurational Monte Carlo moves take on the order of 24 h to complete.

Example 1. For example 1, the simulation system is chosen such that $n = 20$, and $\alpha(T_1)/\alpha(T_n) = 3$. In parallel tempering simulations, it is usual for the temperature to increase exponentially from T_1 to T_n, since higher temperature systems have wider energy histograms, and so higher temperature replicas can be spaced more widely than lower temperature replicas.[12] For specificity, we assume that the wall clock time per configurational step also increases exponentially from $\alpha(T_n)$ to $\alpha(T_1)$. We take N_{move} to be constant for each of the replicas. The allocation of the replicas to the different processors is shown in Figure 2, parts a and b, for the traditional and zero idle time cases, respectively. This example is typical of most parallel tempering simulations that are currently being performed on multiprocessor systems.

Example 2. For example 2, we use n and $\alpha(T_i)$ from example 1. We, furthermore, consider that the correlation times of the lower temperature replicas are longer, and so there should be more configurational moves per Monte Carlo step at the lower temperatures. We consider N_{move} to increase exponentially from $N_{\text{move}}(T_n)$ to $N_{\text{move}}(T_1)$ such that $N_{\text{move}}(T_1)/N_{\text{move}}(T_n) = 100$. With the values for $\alpha(T)$ from example 1, we find $\tau_{\text{long}}/\tau_{\text{short}} = 300$.

Example 3. For example 3, we use $n = 50$, modeling $\alpha(T_i)$ in the same way as in examples 1 and 2, $\alpha(T_1)/\alpha(T_n) = 3$. We model N_{move} in the same way as in example 2, but in this example set $N_{\text{move}}(T_1)/N_{\text{move}}(T_n) = 1000$, since the reason for the increased number of replicas would have been the poor and slow equilibration at the lowest temperatures. We find $\tau_{\text{long}}/\tau_{\text{short}} = 3000$.

4. Discussion

From Table 1, it is clear that the scheduler substantially improves the CPU utilization in parallel tempering simulations. This allows the multiprocessor cluster to be used with confidence, for example, for other jobs or simulations at other parameter values of interest. Example 1 demonstrates that the number of processors used can be reduced by 40% with an increase of only 1.66% in wall clock time. Alternatively, the number of processors can be reduced by 35% and result in no increase in wall clock time relative to the traditional parallel tempering method. As Example 1 is conservative in its characterization of most multiprocessor parallel tempering simulations currently being performed; we anticipate that utilization of the optimal scheduler presented here will result in a large increase in the computational efficiency of parallel tempering simulations.

It is interesting to note that, for all examples, as we increase the number of processors used in the simulations, X, from 1, the wall clock time decreases until the number of processors that result in minimum wall clock time is used, $X^{(n+1)} = \lceil W/\tau_{\text{long}} \rceil$. Increasing the number of processors still further,

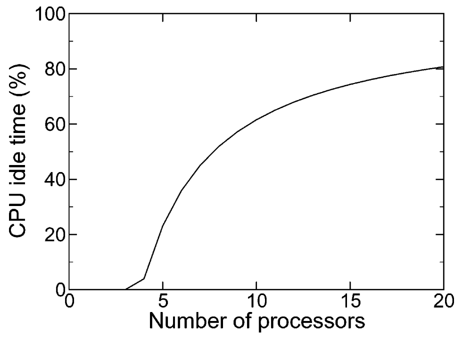

Figure 3. CPU idle time as a function of number of processors used to solve the 20-replica example 2 from section 3.

to say the number of replicas, results in no reduction in overall simulation time and only increases the CPU idle time. This behavior is demonstrated in Figure 3, where the idle time is shown as a function of X for example 2. This figure highlights the importance of proper job scheduling on large, multiprocessor clusters. The use of the optimal scheduler derived here is needed in order for the simulation to make the best use of a large number of CPU cycles. It is theoretically possible to achieve 100% efficiency on multiprocessor systems, making them ideal for parallel tempering simulations. This is especially important in cases where it is desirable to vary N_{move} between different replicas (examples 2 and 3). Taking into account the dependence of the correlation time on temperature is computationally disastrous for the traditional one-replica-per-processor method of performing parallel tempering simulations, as CPU idle times easily become >90%. However, the optimal scheduler makes the simulation of this case feasible, opening the door to performing parallel tempering simulations that sample configurational space more effectively and efficiently.

In the results presented in section 3, we have not explicitly taken into account communication times or the time taken to conduct swap moves. Swap moves that exchange configurations between replicas occur at the beginning of each Monte Carlo step and replica allocations occur at the beginning and possibly once within each Monte Carlo step. These operations are extremely rapid compared to the N_{move} configuration moves performed for each replica, as one can show. Recalling from the Results section that one configurational move takes approximately 0.1 s and knowing that a typical communication time for interprocessor message passing is on the order of 10^{-4} s, we find that example 3 contains the most communication time. In example 3, the increase in idle time due to communication from the zero idle time case is less than 0.00001%. This demonstrates that communication time is not a significant effect in these types of simulations. Communication effects can, thus, safely be ignored.

We have characterized the execution time of each replica in a deterministic fashion, but in reality the execution time is a stochastic quantity due to noise in variables not among the degrees of freedom chosen for the parallel tempering. To model the simulation times more realistically, we have also included randomness into our analysis. That is, the value of α is assumed to fluctuate during each configurational step. As previously mentioned, the accuracy of the estimation of α is dependent on the number of order parameters used to parametrize it. Thus, fluctuations in α will be smaller for systems that use parallel tempering in multiple dimensions. We note that for the case where the temperature is the only parameter used to characterize α, fluctuations in α can be as high as 10−50%. This results in a fluctuation in the time required to complete replica i, which can be represented mathematically as

$$\tau_i = \alpha(T_i)N_{\text{move}}(T_i)\left\{1 + \frac{\sigma}{[N_{\text{move}}(T_i)/\delta(T_i)]^{1/2}}\right\} \qquad (7)$$

where σ is a Gaussian random number, and δ is a value that is proportional to the correlation time. As we generally choose N_{move} to be proportional to the correlation time, we expect N_{move}/δ to be constant. Thus, we use

$$\tau_i = \alpha(T_i)N_{\text{move}}(T_i)[1 + \gamma\sigma] \qquad (8)$$

to model the fluctuations. We examine the cases where $\gamma =$ 0.1, 0.5, and 1.0. To analyze the performance of the scheduler in the presence of the randomness, we take into account that a processor may be idle while it is waiting for another processor to complete its share of calculations on a replica system that is shared between the two processors.

Table 1 shows the results of including randomness into our model for examples 1−3. The averages and standard errors are calculated from the average results from 10 blocks, each containing 1000 runs of the simulation system. The CPU idle time increases monotonically and nonlinearly with γ. For the more complex systems where N_{move} is varied, the inefficiency introduced by the randomness is smaller, since the randomness of several replicas is typically averaged over on most of the processors. The results are encouraging and show that the efficiency of the parallel tempering simulations organized by the scheduler remains within an acceptable limit, even when relatively large fluctuations are considered. Increasing N_{move} will lead to lower fluctuations, with the observed efficiency converging to the $\gamma \rightarrow 0$ limit as $O(1/N_{\text{move}}^{1/2})$.

5. Conclusions

In this paper, we have introduced a theory for the optimal allocation of replicas to processors in parallel tempering simulations. The scheduler leaves intact the balance or detailed balance properties of the underlying parallel tempering scheme. The optimal scheduler derived from the theory allows multiprocessor machines to be efficiently used for parallel tempering simulations. The allocation of replicas to CPUs produced by the scheduler results in a significant enhancement of CPU usage in comparison to the traditional one-replica-per-processor approach to multiprocessor parallel tempering. The optimal scheduling vastly reduces the number of required processors to complete a simulation, allowing an increased number of jobs to be run on a cluster. The computational efficiency of the scheduler also makes it feasible to vary the number of configurational moves per Monte Carlo step, which was not practicable using the one-replica-per-processor scheme, due to the associated large inefficiencies. This flexibility to vary the number of configurational steps is desirable because the correlation time at lower temperatures is often much longer than that at higher temperatures.

Our results show that randomness does not have a significant effect for γ < 0.1, and the performance is still quite tolerable even for the extreme case of $\gamma = 1$. Despite the random execution times, the replica allocation produced by the optimal scheduler is always significantly more efficient than the traditional one-replica-per-processor approach. The idle time caused by random execution times is reduced as the number of configurational moves per Monte Carlo step is increased. Furthermore, parallel tempering in more than one dimension, with order parameters other than temperature, allows for a more accurate determination of the CPU time per replica. For the same reason, these extra dimensions will also aid the sampling efficiency of the underlying parallel tempering algorithm.

Acknowledgment. This work was supported by the U.S. Department of Energy, Office of Basic Energy Sciences.

Appendix

Allocation Scheme for a Heterogeneous Cluster. Using scheduling theory[5,6] it is possible to derive an allocation scheme for a multiprocessor machine with heterogeneous processors. It is assumed that the number of CPU cycles required for each replica to complete one Monte Carlo step and the speed of each of the processors in the machine are known. In this general scheme, the number of processors used by the scheduler, m, is adjusted downward until an acceptably low idle time and total wall clock time are achieved.

For n replicas, where τ_i is the number of CPU cycles required to complete replica i, the total number of CPU cycles required, W, is given in eq 3. We now define

$$W_j = \sum_{i=1}^{j} \tau_i, \quad 1 \le j \le n \tag{A-1}$$

For m processors, where k_i is the speed of each processor in CPU cycles per unit time, with $k_1 \ge k_2 \ge ... \ge k_m$, the total number of CPU cycles available per unit time is

$$K = \sum_{i=1}^{m} k \tag{A-2}$$

We define

$$K_j = \sum_{i=1}^{j} k_i, \quad 1 \le j \le m \tag{A-3}$$

The shortest possible wall clock time to execute the Monte Carlo step for all the replicas is then

$$\tau_{\text{wall}} = \max(W/K, \tau_{\text{long}}) \tag{A-4}$$

where τ_{long} is the maximum value of W_j/K_j, $1 \le j \le m$.

The general scheduler works with a time interval granularity of dt. At the start of the simulation and at the end of each time interval, we assign a level of priority to the replicas. The highest priority is given to the replica with the largest number of CPU cycles required for completion, and the lowest priority is given to the replica with the least number of CPU cycles remaining. A loop is performed through the priority levels, starting at the highest priority. If there are r replicas in the priority level under consideration and s remaining unassigned processors and if $r \le s$, then the r replicas are assigned to be executed on the fastest r number of processors. If the processors have different speeds, each replica must spend an equal amount of wall clock time on each of the processors during the time interval, dt. The total wall clock time for the step is computed from the processor speeds and the required number of CPU cycles. The number of configurational moves that equals $1/r$ of the wall clock time on each processor is computed, and this number is the number of configurational moves that each replica will perform on each processor. For the first $1/r$ of the wall clock time, the replicas are assigned sequentially to the r processors. For the next $1/r$ of the wall clock time, the assignment of the replicas to the processors is cyclically permuted, i.e., replica 1 to processor 2, replica 2 to processor 3, ..., replica r to processor 1. The assignment of replicas to processors is cycled at the end of each $1/r$ of wall clock time until the entire time step is completed. On the other hand if $r > s$, the replicas are assigned to the processors by splitting the time interval in each processor r times, and assigning the replicas to spend one short time interval being processed in each processor. This is accomplished by assigning the first processor to execute sequentially replicas 1, 2, ..., r. The second processor is assigned a cyclic permutation of the replicas to execute sequentially: replicas 2, 3, ..., r, 1. In general processor i executes a cyclic permutation of the replica sequence of processor $i - 1$. This allocation leads to each replica being executed for an equal amount of wall clock time on each processor. A singe replica, moreover, is never allocated to more than one processor at a single point in time.

If there are still processors remaining to be allocated, the replicas at the next lower priority level are allocated by this same process. The procedure is repeated until all processors have been allocated or all replicas have been allocated.

The replica assignment for wall clock time dt is now complete. Replicas are reassigned for the next period of wall clock time using the same rules. If the time interval, dt, is chosen to be small enough, then the total wall clock time of the simulation tends toward τ_{wall}. After the wall clock time of the entire Monte Carlo step has been assigned, the simulation can be performed.

There is some flexibility in the use of this general optimal scheduler for a heterogeneous multiprocessor machine. In general, the best value of m is not known in closed form. It is found by choosing the smallest value of m that gives an acceptably low value of the wall clock time, eq A-4, and an acceptably low idle time in the derived allocation. The time step for the scheduler, dt, must also be chosen. It should be chosen to be small, but not so small that communication effects become significant. Moreover, there must be many configurational Monte Carlo steps per time step, dt, otherwise the splitting of replicas among r processors required by the algorithm will not be possible. The computational time associated with the scheduler will generally be very much smaller than that associated with the simulation. The scheduler may, therefore, be run after each Monte Carlo step. Such use of the scheduler would automatically lead to an adaptive or cumulative estimate of the execution times required by each replica.

In practical application of the results of this general scheduler, the processor allocation will typically be reordered to an equivalent one. For example, in the case of two replicas of equal length to be assigned to a single processor, the algorithm given above will switch between execution of each replica at each time step, dt, rather than complete execution of each replica sequentially. A reordering of the output of the general scheduler, therefore, will generally lead to a simpler processor allocation. Consistent with the constraints of causality, replica execution in time on a single processor may be reordered. Allocation of replicas to processors at each time step, dt, may also be permuted among the processors as along as the idle time so introduced is tolerable.

Alternatively, the schedule optimization for heterogeneous processors can be cast as a linear programming problem. With a penalty for each switch between replicas on a processor, an optimized schedule may be derived at the onset by solving the linear programming problem with a time resolution of dt.

References and Notes

(1) Geyer, C. J. Markov Chain Monte Carlo Maximum Likelihood. In *Computing Science and Statistics: Proceedings of the 23rd Symposium on the Interface*; Keramidas, E. M., Ed.; American Statistical Association: New York, 1991.

(2) Geyer, C. J.; Thompson, E. A. *J. Am. Stat. Assn.* **1995**, *90*, 909.

(3) Marinari, E.; Parisi, G.; Ruiz-Lorenzo, J. Numerical Simulations of Spin Glass Systems. In *Spin Glasses and Random Fields*; Young, A., Ed.; World Scientific: Singapore, 1998; Vol 12.

(4) Manousiouthakis, V. I.; Deem, M. W. *J. Chem. Phys.* **1999**, *110*, 2753.

(5) Coffman, E. G. *Computer and Job-Shop Scheduling Theory;* Wiley: New York, 1976.

(6) Ashour, S. *Sequencing Theory*; Springer-Verlag: New York, 1972.

(7) Scheduler is available under the GPL at http://www.mwdeem.rice.edu/scheduler.

(8) Yan, Q.; de Pablo, J. J. *J. Chem. Phys.* **1999**, *111*, 9509.

(9) Yan, Q.; de Pablo, J. J. *J. Chem. Phys.* **2000**, *113*, 1276.

(10) Faller, R.; Yan, Q. L.; de Pablo, J. J. *J. Chem. Phys.* **2002**, *116*, 5419.

(11) Bunker, A.; Dunweg, B. *Phys. Rev. E* **2001**, *63*, 010902.

(12) Kofke, D. A. *J. Chem. Phys.* **2002**, *117*, 6911.