A Scalable Parallel Monte Carlo Method for Free Energy Simulations of Molecular Systems

MALEK O. KHAN, GARETH KENNEDY, DEREK Y. C. CHAN
Particulate Fluids Processing Centre, Department of Mathematics & Statistics, The University of Melbourne, Parkville, Victoria 3010, Australia

Received 12 May 2004; Accepted 21 July 2004
DOI 10.1002/jcc.20143
Published online in Wiley InterScience (www.interscience.wiley.com).

Abstract: We present a method of parallelizing flat histogram Monte Carlo simulations, which give the free energy of a molecular system as an output. In the serial version, a constant probability distribution, as a function of any system parameter, is calculated by updating an external potential that is added to the system Hamiltonian. This external potential is related to the free energy. In the parallel implementation, the simulation is distributed on to different processors. With regular intervals the modifying potential is summed over all processors and distributed back to every processor, thus spreading the information of which parts of parameter space have been explored. This implementation is shown to decrease the execution time linearly with added number of processors.

 J Comput Chem 26: 72–77, 2005

Key words: parallel computing; high performance computing; Monte Carlo; free energy; molecular simulations

Introduction

Since its conception, the Monte Carlo (MC) algorithm[1–3] has been a major numerical tool for solving problems in the physical sciences.[4] The trends in hardware for high-performance computing is towards parallel commodity processors, either in loosely connected cluster architectures or more integrated shared memory processors, because the performance to cost ratio favors such architectures to purpose-built supercomputers. In this work we introduce a method to effectively deploy MC on parallel computers.

A simple way of utilizing parallel machines is to replace a simulation running for N_{it} iterations on a single processor by spreading the simulation over N_{cpu} processors, each running for N_{it}/N_{cpu}. This type of trivial parallelism always invokes a cost for conventional Metropolis MC because there is a equilibration time involved with every simulation, during which no statistics can be collected.[5] A rule of thumb is that the equilibration time is about 30% of the total simulation time.[2] Because equilibration needs to be run for every independent process, this overhead cannot be eliminated by this trivial method of parallelization.

Parameter or data parallelism, in which different sets of parameters are run on different processors, is another form of trivial parallelism. It suffers from the obvious drawback that N_{cpu} different parameter sets are needed. Many problems will only involve on the order of 10 such value sets, thus rendering it impossible to effectively perform a parameter parallel calculation with a large amount of CPUs. Also, for large systems, there can be a considerable period of waiting time before good statistics can be accumulated, rendering the project cumbersome.

Domain decomposition is a common parallel method, in which particles are divided on different processors according to their geometrical positions.[5] This approach is not suitable for systems with long-range interactions, for example, Coulomb interactions, because no cutoffs can be employed. Parallel expanded ensembles is often used for problems where the energy dominates the entropy. An example is parallel tempering, in which simulations are performed at different temperatures to speed up the convergence of the lowest temperature system.[6,7]

In this report we present a general way to effectively parallelize a Monte Carlo algorithm that also gives the free energy of the system as a direct output of the simulation. Traditional Metropolis MC samples phase space weighted by the Boltzman probability distribution $p \sim \exp(-U/kT)$, where k is the Boltzmann constant, T is the temperature, and U is the Hamiltonian of the system. In the canonical ensemble, conventional simulations do not give the free energy (or the underlying partition function) directly, but rather derivatives of the free energy. The most straightforward way to obtain the free energy is to perform several simulations while

Correspondence to: M. O. Khan; e-mail: m.khan@unimelb.edu.au

Contract/grant sponsors: Australian Research Council, the Particulate Fluids Processing Centre at the University of Melbourne, The Victorian Partnership for Advanced Computing and the Wenner–Gren Foundation

varying one parameter (e.g., the temperature) and then integrate over that parameter, a method commonly referred to as thermodynamic integration. More elegant simulation schemes used to obtain the free energy include thermodynamic perturbation,[8] umbrella sampling[9] and expanded ensembles.[10]

Recently, so-called flat histogram techniques have evolved as a technique to calculate the free energy directly in a MC simulation.[11–18] Flat histogram techniques resemble umbrella sampling in that an external potential is added to the system energy. In contrast to umbrella sampling the external potential is not given as an input to the simulation but is modified throughout the simulation to achieve an equal probability (flat histogram) of visiting all values of any chosen parameters. At the end of the simulation the external potential is directly related to the free energy. As with umbrella sampling this method is most useful for systems that have a complex free energy landscape, that is, for systems where traditional Boltzmann guided sampling would lead to extremely long convergence time.

One proposed method of parallelizing simulations utilizing the free energy method is to impose artificial constraints and manually allocate the required parameter space onto different processors.[13] Although this could be effective in some cases, it would put restraints on the use of effective global moves, which is known to be able to speed up simulations considerably. One example of a global move is the pivot rotation commonly used in polymer simulations.[19,20] Also, the different regions of parameter space need a certain degree of overlap to fit the simulations together, and this may induce a substantial overhead when many processors are used.

The concept introduced in this report is to let every processor perform calculations in any region of parameter space. With regular intervals the processors communicate and the algorithm decides where more calculations need to be performed. This is propagated out to the processors. A weight function (histogram) directs the calculations towards the areas of parameter space that need to be explored. Fixed parameter boundaries are not given in which every processor performs its calculations; rather, the calculation of a global (over all the processors) weight function ensures that the parallelization is done effectively.

To illustrate the method we have chosen to calculate the free energy as a function of the end-to-end distance of charged polymers. After a description of the model and method, we report results of parallel simulations scaling up above 32 processors.

Simulation Method

The polymer model is a simple bead-and-stick model with N_{mon} hard sphere monomers, with a diameter of 4 Å connected by fixed length bonds (6 Å). To test the model on a complex system, charges are added to the polymer. For the charged polymer, known as a polyelectrolyte, every monomer has a charge of $+1$, and N_c hard sphere counterions of the same size as the monomers, with a charge -3 are included to keep the system electroneutral. Trivalent counterions interact strongly with the charged polymer backbone and will collapse the polymer. This feature makes the simulation converge slowly when conventional MC is used. All particles are enclosed in a spherical cell and the electrostatic interactions are calculated with Coulomb's law. The total electrostatic energy is given by

$$U = \sum_{i<j}^{N_{\mathrm{mon}}+N_c} \frac{q_i q_j}{4\pi\varepsilon_0\varepsilon|\mathbf{r}_i - \mathbf{r}_j|} \tag{1}$$

where $|\mathbf{r}_i - \mathbf{r}_j|$ is the distance between particles i and j, q_i is the charge of particle i, ε_0 is the dielectric permittivity of vacuum and ε is the relative dielectric constant of the solvent. In this so-called primitive model the solvent only enters by screening the electrostatic interaction by ε. Here we use the value for water, $\varepsilon = 78$.

The polymer is moved with a pivot move[19,20] and the small ions by simple translation. The simulations are carried out in the canonical ensemble, and all moves are accepted according to the normal Metropolis MC rules.[1–3] Here we define one MC iteration as $10 \times N_{\mathrm{mon}}$(1 pivot + $N_{\mathrm{mon}}/2$ translations). As a measure of the polymer conformation, the end-to-end distance R_{ee}, defined as the distance between the first and last monomer on the chain, is used.

A straightforward way of calculating the free energy or potential of mean force (PMF), $w(R_{ee})$, in a normal simulation, is to simply calculate the probability of finding the system at a certain end-to-end distance $p(R_{ee})$, because the PMF is related to this probability by

$$w(R_{ee}) = -kT \ln p(R_{ee}). \tag{2}$$

But because the probability of visiting high energy states is low, configurations far from the average R_{ee}, that is, the extended or compressed configurations, will be sampled infrequently if at all during a simulation. However, by adding an appropriate penalty function U^* to the normal, undisturbed Hamiltonian U it is possible to sample all states of interest with equal probability.[11,12] Generating a uniform probability results in[11,12,18]

$$w(R_{ee}) = -U^*(R_{ee}) + C \tag{3}$$

where C is a physically unimportant constant setting the zero level of the free energy. More details of the model and the serial algorithm can be found in earlier work concerning polyelectrolyte behavior.[18]

To construct a U^* that will give rise to an uniform distribution of end-to-end distances the function $U^*(R_{ee})$ is discretized over R_{ee} into equal size intervals. The number of bins used is between 100 and 1000. At the start of the simulation the penalty function U^* is taken to be uniform. Every time the end-to-end distance falls within a particular interval of R_{ee} the corresponding U^* is increased by a certain value δU^*. This ensures that the distribution function $p^* \sim \exp[-\beta(U + U^*)]$ will approach a constant.[11,12,18]

A technical problem is that keeping δU^* constant during the full extent of the simulation will lead to poor statistics. The difference in energy before and after a MC move will be calculated as $\Delta U + \Delta U^*$. When the distribution function becomes uniform, U^* will start to be updated equally over all R_{ee}, and the fluctuations in ΔU^* will dominate and obscure the information in ΔU. This problem is circumvented by decreasing the penalty as the

simulation progresses.[11,13] In this article the choice for δU^* is $0.1kT$ to $0.001kT$ at the beginning of the simulation. Following the prescription of Wang and Landau[13] the simulation is run until $p^*(R_{ee})$ is "flat," when δU^* is updated according to $\delta U^*_{\text{new}} = \delta U^*/2$. "Flat" is defined by all values of $p^*(R_{ee})$ being within δp from the mean $\langle p^*(R_{ee})\rangle$, where δp is between 0.1 to 0.35 depending on the complexity of the system. When δU^* is updated the histogram containing $p^*(R_{ee})$ is reset to zero and a new histogram is collected for the new value of δU^*. The simulation goes on until δp reaches a predecided number, normally between 10^{-5} and 10^{-8}.

In the parallel version of the free energy algorithm, N_{cpu} processors run identical versions of the program but with different initial configurations (actually only the initial random number seeds are different). Every process runs independently except that at certain intervals the processor summed PMF, $\sum_{i=1}^{N_{\text{cpu}}} w_i(R_{ee})$, is distributed to all processors. Each process then continues independently, but with the global PMF. The idea is that every process does not have to explore the full PMF as a function of R_{ee}, but together they will, because every now and then the processes tell each other which R_{ee} they have visited.

In the same manner the processor averaged distribution function

$$\langle p^*(R_{ee})\rangle_{N_{\text{cpu}}} = \frac{1}{N_{\text{cpu}}} \sum_{i=1}^{N_{\text{cpu}}} p_i^*(R_{ee}) \tag{4}$$

is gathered and this average is checked against the flatness criteria.

The interval between interprocessor communication is an input to the simulation. If the interval is of the same size as the time it takes to run the total simulation, the different processors will not be connected, resulting in N_{cpu} independent simulations. Obviously, communicating too often will slow down the simulation. In this work we report on results where communication is performed every MC iteration (as defined above). The choice is motivated from the structure of the original serial program used, for which it is suitable to communicate at any multiple of an MC iteration. Initial testing of the system used here showed that the execution time did not vary with this multiple down to one MC iteration. Of course, for smaller problems, where less time is spent in every MC iteration, the interval between communication may have to be increased, which can effect the resulting efficiency of the algorithm. In any case, the method is envisaged to be used for large systems where a single CPU cannot meet the computational demand in reasonable time.

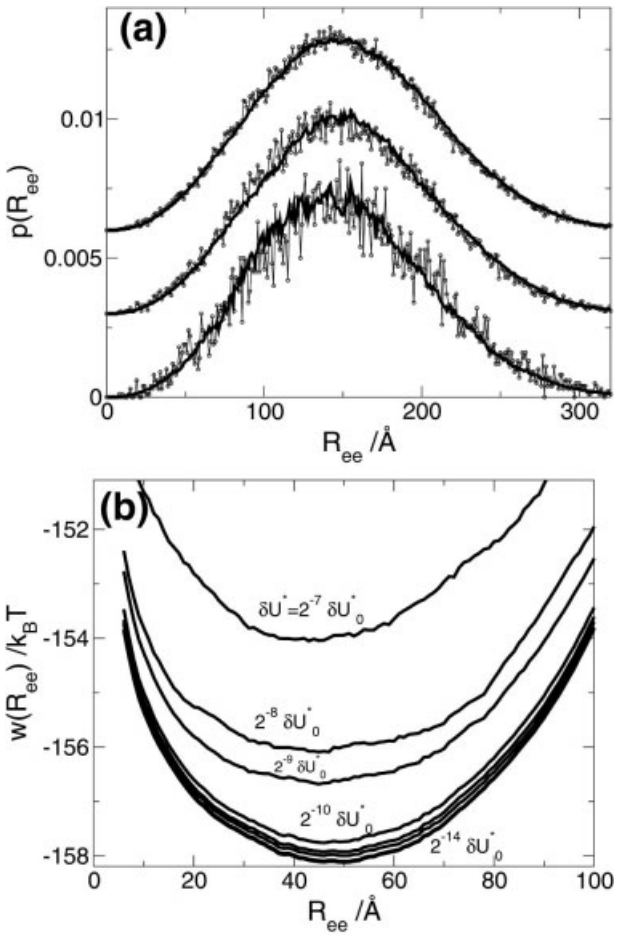

Figure 1. (a) The probability distribution function of the end-to-end distance R_{ee} of an uncharged polymer with $N_{\text{mon}} = 240$. The thick solid curves are from free energy simulations (via the PMF) run on $N_{\text{cpu}} = 16$ processors, and the thin fluctuating curves are from conventional MC. The curves that are on top of each other are run for the same number of iterations (i.e., every processor runs 1/16 of the iterations in the parallel algorithm). From top to bottom the number of iterations are 8, 4, and 1×10^6. The probability curves for the first two cases have been displaced vertically for visual clarity. (b) The potential of mean force as a function of R_{ee} for a polyelectrolyte with $N = 63$ and trivalent counterions. The different curves are the PMFs when the end-to-end distribution is deemed flat with a certain δU^*. The original update is $\delta U_0^* = 0.001kT$ and the simulation is run until $\delta U^* = 2^{-14}\delta U_0^* = 6 \times 10^{-8}kT$. This simulation was run on $N_{\text{cpu}} = 16$ processors. Only the last eight updates are shown.

Results and Discussion

First, the results using the parallel PMF method is compared with conventional MC for the case of an uncharged polymer. In Figure 1a the distribution of end-to-end distances, $p(R_{ee})$ is calculated from the PMF using eq. (2) and compared with results from the same simulation without adding a penalty, where $p(R_{ee})$ is calculated directly, in the standard way. The two methods lead to distribution functions with the same shape. The PMF method is very effective in comparison with conventional MC methods near phase separation, because conventional MC cannot sample over phase space with high free energy barriers, but even for the simple case of an uncharged hard sphere polymer, the PMF method is found to be useful.

To clarify how the PMF evolves in the simulation, the PMF of a polyelectrolyte is plotted in Figure 1b at updates of δU^*. At the early stages of the simulation, for large δU^*, the PMF takes on the correct form on a large length scale but depicts some fluctuations. For decreasing δU^* the curve becomes progressively smoother

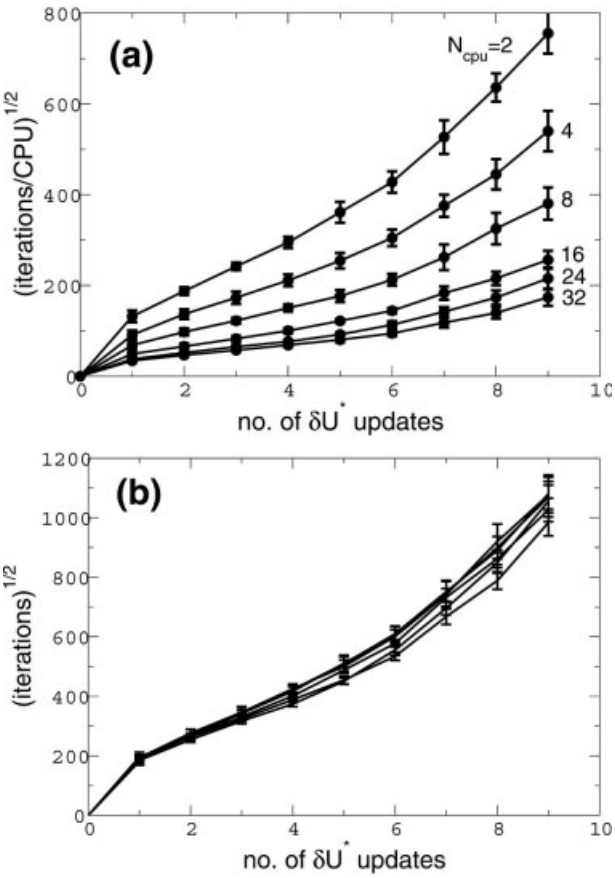

Figure 2. (a) The number of iterations per CPU before an update of δU^* is made. For each N_{cpu}, Eleven simulations have been performed and the mean is shown. The error bars show the standard deviation. It should be noted that the lines are just a guide to the eye, because δU^* is actually constant between the points. (b) Same as (a) except that the y-axis now shows the total number of iterations before an update of δU^* is made. The curves for different N_{cpu} have not been labeled, because they all are on top of each other within the statistical errors. The system simulated is the same as accounted for in Figure 1b.

because the crude form of the PMF created by using a large δU^* is filled in by using smaller δU^*. The distance between the curves in Figure 1b, corresponding to $\delta U^* = 2^{-7} \times 0.001kT$ and $\delta U^* = 2^{-8} \times 0.001kT$, is around $2kT$. This means that about 200 iterations/bin are performed before the update. The number of iterations needed before the flatness criteria is reached and δU^* is updated is shown in Figure 2, and it is evident that the smaller the δU^*, the longer time it takes to reach the flatness criteria.

The update of δU^* is a measure of how fast the simulation converges because the update only occurs when $U^*(R_{ee})$ has evolved enough to give a flat $p^*(R_{ee})$. To illustrate the efficiency of the parallel algorithm, Figure 2 shows the updates as a function of number of MC iterations for simulations with different number of processors. To collect statistical material, simulations have been run 11 times for every N_{cpu}.

In Figure 2a the number of iterations on every processor needed before $p^*(R_{ee})$ is flat enough to allow for an update is depicted. It is clear that when more processors are used, the faster this criteria is reached. In Figure 2b the total number of iterations needed before an update of $p^*(R_{ee})$ is shown to be the same for $2 \leq N_{\text{cpu}} \leq 32$, thus running the problem on 32 processors completes the simulation with 16 times less iterations/processor than running the simulation on two processors.

In the original serial algorithm, every bin has to be visited an equal amount of times, within the flatness criteria, and the single processor has to cover the full available parameter space. In the parallel algorithm only the total, processor averaged, distribution function has to be flat. When examining the distribution functions calculated by the individual processors, these are far from flat as displayed in Figure 3.

During the simulation, information is passed on between the processors of which values of the parameter space needs to be further explored, via the processor summed $U^*(R_{ee})$. When only a few processors are employed, (Fig. 3a), all processors cover a large part of the parameter space, while for many processors (Fig.

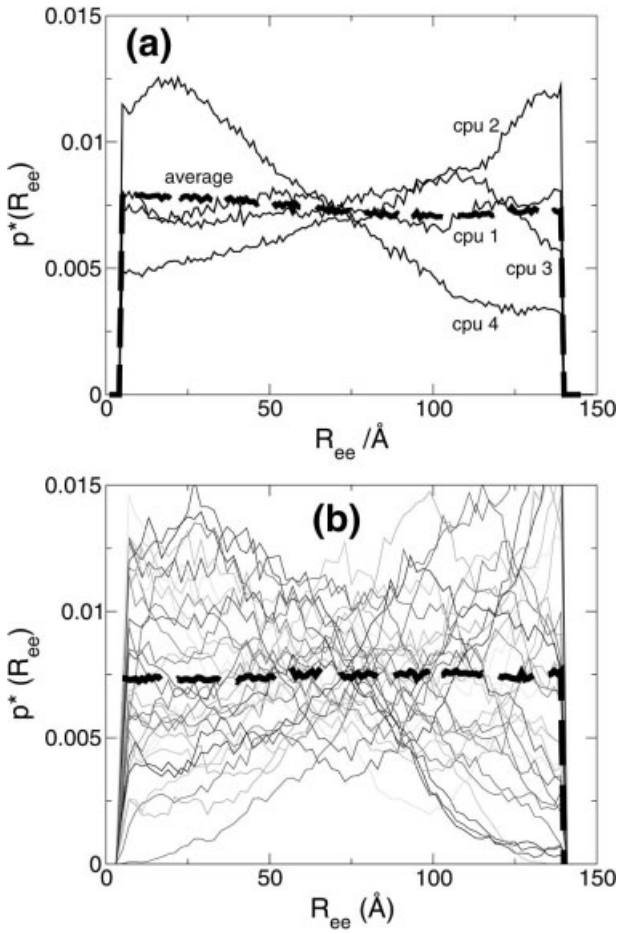

Figure 3. The distribution functions from individual processors in a parallel simulation (thin solid lines) and the average over all processors (thick dashed line). (a) $N_{\text{cpu}} = 4$ and the different lines are labeled. (b) $N_{\text{cpu}} = 32$, here the multitude of different lines prohibit labelling. The system simulated is the same as accounted for in Figure 1b.

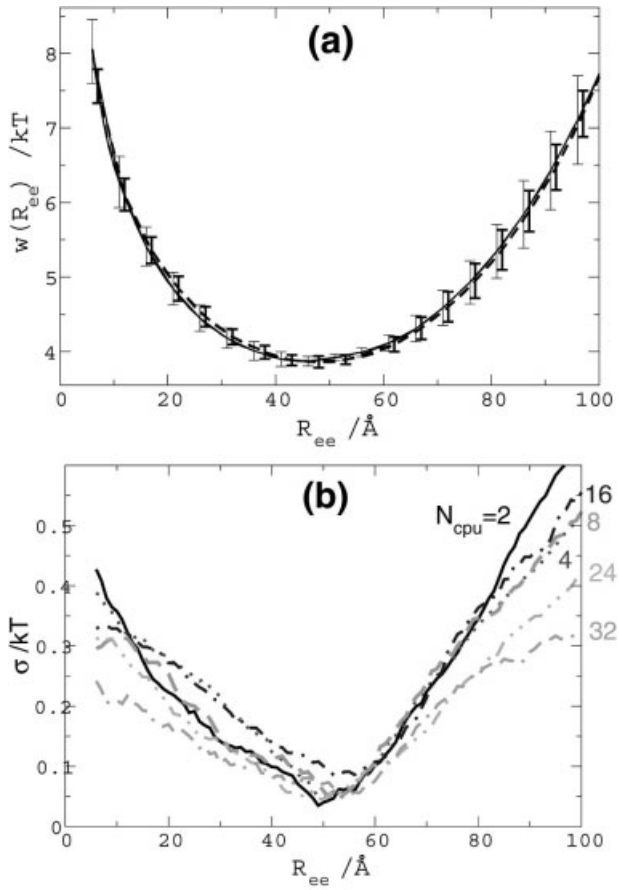

Figure 4. (a) The PMF, with error bars corresponding to the standard deviation calculated from 11 independent simulations, for $N_{cpu} = 2$ (solid line with thin error bars) and $N_{cpu} = 32$ (dashed line with thick error bars). (b) The standard deviation, calculated from 11 independent simulations, for different N_{cpu}. The system simulated is the same as accounted for in Figure 1b.

3b) the processors can be more specialized only covering a small part of the parameter space, still resulting in a flat processor averaged $\langle p^*(R_{ee})\rangle_{N_{cpu}}$.

Although the convergence criteria is meet with the same number of total iterations, independent of N_{cpu}, the question remains if the different simulations produce results of the same quality. For conventional MC it is possible to estimate statistical errors by collecting independent values of the sought after parameter during the simulation. For flat histogram techniques this it not applicable because it is only the end product that gives the wanted solution. The flatness criteria is a guide to how the simulation progresses, but not a measure of the quality of the result.

To quantify the errors of the parallel simulations, we have run simulations 11 times for each value of N_{cpu}, each of the 11 simulations having different initial configurations. Using this experimental approach it is possible to calculate standard deviations for the PMF as shown in Figure 4. Because there always is ambiguity concerning the zero-level of the PMF, the PMF has first been converted to the distribution function $p(R_{ee})$, which is then normalised, $\Sigma_{R_{ee}} p(R_{ee}) = 1$, and converted back to a potential.

Table 1. The Standard Deviations of the PMF, as Defined in Figure 4a, for Certain End-to-End Distances R_{ee}.

R_{ee} (Å)	10	20	30	40	50	60	70	80	90
σ_2 (kT)	0.36	0.22	0.14	0.10	0.04	0.10	0.22	0.34	0.51
σ_{32} (kT)	0.21	0.17	0.11	0.08	0.07	0.08	0.19	0.25	0.29

σ_2 is the standard deviation calculated when two processors are used and σ_{32} is the corresponding standard deviation for 32 processors.

In Figure 4a the PMF is shown for the two cases of $N_{cpu} = 2$ and $N_{cpu} = 32$. Within the standard deviation the two results are equal. Also, the figure shows that the standard deviation for the two cases are of equivalent size. A few of the standard deviations are also shown in Table 1. To further compare the standard deviations, these are shown as a function of the end-to-end distance in Figure 4b. From this data we conclude that independent of the number of processors, the simulations produce results of equal quality.

Even though the number of iterations/processor needed to complete the simulations decrease linearly with added processors, it does not necessarily mean that the simulations runs equally fast. It is obvious that for every problem there is a number of processors that eventually will make the parallelisation inefficient due to communication overhead. The relatively small problem of a 63 monomer polyelectrolyte with 21 trivalent counterions, scales up to about 32 processors on a cluster type machine as seen in Figure 5. In the implementation used, communication is performed after every iteration. This is often enough to maintain the N_{cpu} independent convergence shown in

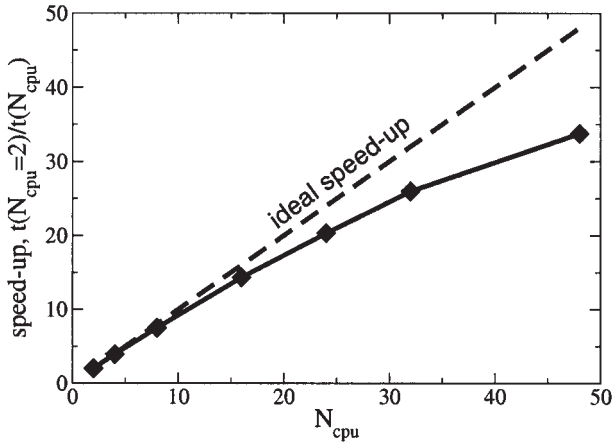

Figure 5. The speed-up, in comparison with the $N_{cpu} = 2$ case, found by using the parallel method. The curve is normalized to 2 for $N_{cpu} = 2$. The system simulated is the same as accounted for in Figure 1b and the machine is the Victorian Partnership of Advanced Computing 97 node, 194 CPU Linux Cluster based on Xeon 2.8 GHz CPUs with a Myrinet interconnect.

Figure 2, but infrequent enough to not slow down the simulation. Further trials may show that communication can be made less frequently, which would be important when using a large amount of processors. In any case, for larger and more complex problems, we anticipate that this method will scale to far more processors, because more calculations are performed in between processor communication.

Conclusion

We have shown how the newly introduced flat histogram MC method,[11,12] which is used to directly obtain the free energy of a system, can be efficiently parallelized by distributing the calculation of the modifying potential U^* on to multiple processors. By doing this, every processor only explores a part of parameter space, while all processors combined visit the full parameter space with equal probability (see Fig. 3). The method converges with the same number of iterations (Fig. 2), and gives results with the same error (Fig. 4), independent of N_{cpu}. The overhead for adding extra processors to the calculation is small, (see Fig. 5), thus leading to a linear decrease of the execution time with added processors.

References

1. Metropolis, N. A.; Rosenbluth, A. W.; Rosenbluth, M. N.; Teller, A.; Teller, E. J Chem Phys 1953, 21, 1087.
2. Allen, M. P.; Tildesley, D. J. Computer Simulation of Liquids; Oxford University Press: Oxford, 1989.
3. Frenkel, D.; Smit, B. Understanding Molecular Simulation; Academic Press: San Diego, 1996.
4. Dongarra, J.; Sullivan, F. Comput Sci Eng 2000, 2, 22.
5. Heffelfinger, G. S.; Lewitt, M. E. J Comput Chem 1996, 17, 250.
6. Hansmann, U. H. Chem Phys Lett 1997, 281, 140.
7. Irbäck, A.; Sandelin, E. J Chem Phys 1999, 110, 12256.
8. Zwanzig, R. W. J Chem Phys 1954, 22.
9. Torrie, G. M.; Valleau, J. P. J Comp Phys 1977, 23, 187.
10. Lyubartsev, A.; Martsinovski, A. A.; Shevkunov, S. V.; Vorontsov–Velyaminov, P. N. J Chem Phys 1992, 96, 1776.
11. Engkvist, O.; Karlström, G. Chem Phys 1996, 213, 63.
12. Wang, F.; Landau, D. P. Phys Rev Lett 2001, 86, 2050.
13. Wang, F.; Landau, D. P. Phys Rev E 2001, 64, 056101.
14. Kim, E. B.; Faller, R.; Yan, Q.; Abbott, N. L.; de Pablo, J. J. J Chem Phys 2002, 117, 7781.
15. Yan, Q.; Faller, R.; de Pablo, J. J. J Chem Phys 2002, 116, 8745.
16. Hansmann, U. H.; Wille, L. T. Phys Rev Lett 2002, 88, 068105.
17. Shell, M. S.; Debenedetti, P. G.; Panagiotopoulos, A. Z Phys Rev E 2002, 66, 056703.
18. Khan, M. O.; Chan, D. Y. C. J Phys Chem B 2003, 107, 8131.
19. Lal, M. Mol Phys 1969, 17, 57.
20. Madras, N.; Sokal, A. D. J Stat Phys 1988, 50, 109.