A scalable parallel Monte Carlo method for free energy simulations of molecular systems

Malek O. Khan[1], Gareth Kennedy[2],, and Derek Y. C. Chan[2]

[1] Uppsala University, Dept. of Physical Chemistry, Box 579, 751 23 Uppsala, Sweden
malek.khan@fki.uu.se,
WWW home page: http://www.anst.uu.se/~malkh812/
[2] The University of Melbourne, Department of Mathematics & Statistics, Parkville, Victoria 3010, Australia

Abstract. We present a method of parallelising 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 which 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. ...

1 Introduction

In this paper we present a general way to effectively parallelise a Monte Carlo (MC) algorithm which also gives the free energy of the system as a direct output of the simulation. Traditional Metropolis MC samples phase space weighted by the Boltzmann 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. Recently, so-called flat histogram techniques have evolved as a technique to calculate the free energy directly in a MC simulation [1–3]. 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 which have a complex free energy landscape, i.e. for systems where traditional Boltzmann guided sampling would lead to extremely long convergence time.

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) histogram ensures that the parallelisation 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.

2 Simulation method

The polymer model is a simple bead and stick model with N_{mon} hard sphere monomers, with a diameter of 4Å connected by fixed length bonds (6Å), see reference for further technical details. The polymer is moved with a pivot move 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. Here we define one MC iteration as $10 \times N_{mon}$(1 pivot+$N_{mon}/2$ 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 straight-forward 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 $p(R_{ee})$, since the PMF is related to this probability by $w(R_{ee}) = -kT \ln p(R_{ee})$. But since the probability of visiting high energy states is low, configurations far from the average R_{ee}, i.e. 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 [1, 2]. Generating a uniform probability results in [1–3] $w(R_{ee}) = -U^*(R_{ee}) + C$, 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 behaviour [3].

In order 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 [1–3]. Following the prescription of Wang and Landau [2] the simulation is run until $p^*(R_{ee})$ is "flat", when δU^* is updated according to $\delta U^*_{new} = \delta U^*/2$.

In the parallel version of the free energy algorithm, N_{cpu} processors run identical versions of the program but with different initial configuration (actually only the initial random number seed is different). Every process runs independently except that at certain intervals the processor summed PMF, $\sum_{i=1}^{N_{cpu}} w_i(R_{ee})$, 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 since every now and then the processes tell each other which R_{ee} they have visited.

In the same manner the processor averaged distribution function $\langle p^*(R_{ee})\rangle_{N_{cpu}}$ is gathered and this average is checked against the flatness criteria.

3 Results and discussion

The update of δU^* is a measure of how fast the simulation converges since 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 1 shows the updates as a function of number of MC iterations for simulations with different number of processors. In order to collect statistical material, simulations have been run 11 times for every N_{cpu}.

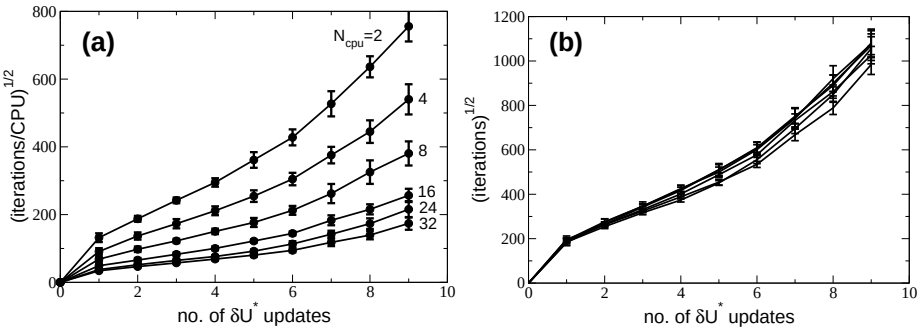

Fig. 1. (a) The number of iterations per CPU before an update of δU^* is made. For each N_{cpu}, 11 simulations have been performed and the mean is shown. The error bars show the standard deviation. It should be noted that the lines are just a guide to the eye, since δU^* is actually constant between 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 labelled, since they all are on top of each other within the statistical errors. The system simulated is a polyelectrolyte with $N_{mon} = 63$ and trivalent counterions.

It is clear that when more processors are used, the faster this criteria is reached. In Figure 1b the total number of iterations needed before an update of

$p^*(R_{ee})$ is shown to be the same for $2 \leq N_{cpu} \leq 32$, thus running the problem on 32 processors completes the simulation with 16 times less iterations/processor than running the simulation on 2 processors.

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 2. In the implementation used, communication is performed after every iteration. This is often enough to maintain the N_{cpu} independent convergence shown in Figure 1, 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, since more calculations are performed in between processor communication.

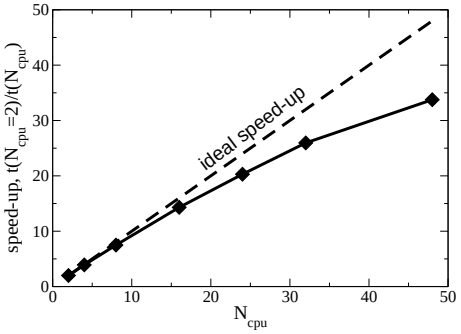

Fig. 2. The speed-up, in comparison with the $N_{cpu} = 2$ case, found by using the parallel method. The curve is normalised to 2 for $N_{cpu} = 2$. The system simulated is the same as accounted for in Figure 1 and the machine is the Victorian Partnership of Advanced Computing 97 node, 194 CPU Linux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

References

1. Engkvist, O., Karlström, G.: A method to calculate the probability distribution for systems with large energy barriers. Chem. Phys. **213** (1996) 63–76
2. Wang, F., Landau, D.P.: Efficient, multiple-range random walk algorithm to calculate the density of states. Phys. Rev. Lett. **86** (2001) 2050–2053
3. Khan, M.O., Chan, D.Y.C.: Monte Carlo simulations of stretched charged polymers. J. Phys. Chem. B **107** (2003) 8131–8139