Parallel Prefetching for Canonical Ensemble Monte Carlo Simulations

Harold W. Hatch*

Chemical Informatics Research Group, Chemical Sciences Division, National Institute of Standards and Technology, Gaithersburg, Maryland 20899-8380, USA

E-mail: harold.hatch@nist.gov

Abstract

In order to enable large-scale molecular simulations, algorithms must efficiently utilize multi-core processors that continue to increase in total core count over time with relatively stagnant clock speeds. Although parallelized molecular dynamics (MD) software has taken advantage of this trend in computer hardware, single-particle perturbations with Monte Carlo (MC) are more difficult to parallelize than system-wide updates in MD using domain decomposition. Instead, prefetching reconstructs the serial Markov chain after computing multiple MC trials in parallel. Canonical ensemble MC simulations of a Lennard-Jones fluid with prefetching resulted in up to a factor of 1.7 speedup using 2 threads, and a factor of 3 speedup using 4 threads. Strategies for maximizing efficiency of prefetching simulations are discussed, including the potentially counter-intuitive benefit of reduced acceptance probabilities. Determination of the optimal acceptance probability for a parallel simulation is simplified by theoretical prediction from serial simulation data. Finally, complete open-source code for parallel prefetch simulations was made available in the Free Energy and Advance Sampling Simulation Toolkit (FEASST).

1 Introduction

With the emergence of exascale high-performance computing and the increase in number of processor cores over time, but a relatively stagnant central processing unit (CPU) clock speed, development and distribution of parallel algorithms to the scientific community is required to enable next-generation molecular simulations on a scale that is not currently possible. But parallelization of Markov chain Monte Carlo (MC)[1] is hindered by local perturbations, as opposed to global molecular dynamics (MD) time steps, and the apparent requirement to maintain detailed balance to ensure ergodicity,[2,3] although strict detailed balance may be unnecessary.[4] While some are tempted to replace MC in favor of parallel MD when possible, MC is capable of performing calculations that MD is not.[3,5,6] Hybrid MC/MD is a promising technique that simultaneously takes advantage of the parallelization of MD and the unique MC statistical sampling algorithms; however, hybrid MC/MD must be implemented carefully to ensure detailed balance.[7–9] This work focuses upon parallelization of MC for molecular simulations.[10]

Examples of MC parallelization include prefetching,[11–14] domain decomposition[15–20] and order parameter decomposition.[21,22] Monte Carlo algorithms that improve serial efficiency but are also parallelizable include configurational bias,[23–25] waste recycling,[26] event-chain[27,28] and efficient pair-interaction schemes[6] including modified cell-linked list[29] and k-d tree search.[30,31] To begin, prefetching is described. The advantages, disadvantages and potential synergy between prefetching and the other parallelization methods are then discussed in the above order.

Prefetching performs a parallel batch of simultaneous trial perturbations from the same initial state, and then reconstructs the Markov chain as if the parallel batch of trials were performed in serial.[11–14] A long simulation is comprised of many parallel batches. During reconstruction of the serial Markov chain, rejected trials revert back to the initial state. Thus, if a trial is rejected, the next trial already computed in the parallel batch may be considered. In contrast, accepted trials produce a new state. Thus, any subsequent trials

already computed in the parallel batch based on the initial state cannot be used, and the computational time spent on these trials is wasted. This wasted computational effort can only be minimized by lowering the probability to observe an accepted trial in a given batch of parallel threads. But lower acceptance probability reduces statistical sampling efficiency. Thus, a disadvantage of prefetching is that trials with higher acceptance probability lower the parallel efficiency.

Domain decomposition divides space into independent regions that can be perturbed in parallel. This approach is efficient when interaction ranges are many times less than the system size. On the other hand, reduced system sizes may be preferred in some applications. For example, grand canonical ensemble simulations of self-assembly[32,33] may minimize system size effects compared to the canonical ensemble. Also note that domain decomposition can be used in conjunction with prefetching. If prefetch batches are divided into independent domains, then an accepted trial in one domain no longer invalidates other trials in independent domains.

Order parameter decomposition in flat-histogram methods[21,22,34] refers to dividing a large order parameter range into a number of subintervals. For example, in a grand canonical ensemble flat-histogram Monte Carlo simulation, where number of particles is the order parameter, the particle range of $[0, 600]$ could be decomposed into two simulations with $[0, 305]$ and $[295, 600]$ particles, respectively. In this example, there are 11 overlapping order parameter states. More overlap facilitates obtaining a smooth free energy curve over the entire range, but at the expense of redundant simulation compared to the serial approach. Scaling over many processors can be limited due to reduced statistical sampling when a single simulation does not span the transition between two (meta)stable structures.[35] For example, temperature expanded ensembles of structural transitions between rounded squares and rods were inefficient to decompose by order parameter, but serial simulations took up to a month to converge (i.e., 15 Wang-Landau flatness conditions).[36,37] The order parameter range can also be decomposed into individual states. In this case, ghost insertions and deletions make

the simulations embarrassingly parallel because the states are independent.[22] The efficiency with ghost trials would only lessen, in comparison to a serial grand canonical simulation, when insertions and deletions contribute significantly to the statistical sampling efficiency. This may lead to a trade-off between parallelization efficiency and sampling efficiency when order parameter ranges are decomposed for increasing number of processors. Note that order parameter decomposition can be used in conjunction with prefetching, domain decomposition or both.

Configurational bias (CB) Monte Carlo is another technique that is parallelizable.[23–25] CB is often used to efficiently grow or move atoms in molecules by breaking a trial into serial steps, with each step computing multiple, independent trial positions of an atom. Each step in the CB algorithm may be parallelized by assigning the energy computation of the independent trial positions to each thread. In addition, dual-cut CB increases the efficiency of the simulations by considering only the fast, short-range part of the potential (e.g., excluded volume) during these steps, and computing only the full, expensive potential (e.g., charges or Lennard-Jones tails) at the end.[38] But when dual-cut CB is utilized, parallelization efficiency is diminished because the computational time spent on the full, expensive potential may not be much less than the time spent on the fast, short-range potential. Parallel configurational bias can also be used in conjunction with prefetching, domain decomposition and order parameter decomposition.

Waste recycling[26] utilizes the information of rejected states and is parallelizable with the benefit of selecting the parallel trial by Boltzmann weight without the burden of computing the weights of the old state, as in CB. However, users may be wary of the additional complexity of computing ensemble averages and the violation of superdetailed balance. While waste recycling may be more efficient than prefetching, the study of optimized acceptance probabilities for prefetching are also applicable to parallel CB and waste recycling.

This work considers one variant of prefetching with a single-particle displacement trial in the canonical ensemble. Cases where the computational time required to perform one trial is

very different from another are not considered. Each parallel thread completes one trial and waits for the other threads to complete their trials. In many ways, this prefetching variant is similar to a parallel one-step CB algorithm for single-site spherical particles. However, CB is more likely to select the most favorable of multiple perturbations that are likely to be accepted, rather than the arbitrarily-assigned first accepted trial. But CB requires computation of both the new and old configurations to obtain the acceptance criteria.[3] While approximately half the computer time is spent on the old configuration in canonical ensemble CB, prefetching does not have this burden because it only considers new configurations.

Prefetching, parallel one-step CB and parallel waste recycling also suffer from a similar parallel overhead. This overhead cost varies inversely with the computational time required to calculate the energy change of the new perturbation, and varies directly with the computational time required to perform the perturbations and synchronize the parallel threads. The efficiency of prefetching is thus governed by the following three variables: the overhead, the trial acceptance probability and the number of parallel threads. A range of all three of these variables are studied so that others may more easily optimize their own parallel prefetch simulations. The optimizations presented here are also relevant to parallel CB and parallel waste recycling.

A single component Lennard-Jones fluid is a sufficient benchmark system to study the efficiency of prefetching. This is because the efficiency is solely determined by the overhead, acceptance and number of threads. With these three variables in mind, the potential function simply determines the CPU time required to compute the energy of a single particle. Additional benchmarks with more complex models would increase the energy computation time relative to the computation time required to perform the perturbation and synchronize the threads (e.g., decrease overhead and increase efficiency). Potential optimizations such as neighbor and cell lists are also not utilized in this work. These optimizations would only affect the overhead, by reducing the energy computation time and possibly increasing the time to synchronize threads. Strategies to reduce synchronization time will be discussed.

An appropriate measure of efficiency depends upon the type of Monte Carlo simulation and the resulting properties of interest. Energy fluctuations,[39] mean squared displacement and diffusion coefficients[40] have been shown to be appropriate metrics for the efficiency of Monte Carlo simulations in the canonical ensemble.[41,42] In this work, efficiency is defined based on the diffusion coefficient with the time scale of total real-time CPU-hours. The literal interpretation of this efficiency metric is the rate at which a simulated particle diffuses from its initial position as observed by the simulator, and is thus not only dependent on the simulated material but also the computer hardware and software implementations. A more simple efficiency metric based on the relative number of Monte Carlo trials performed per real-time CPU-hours is also reported.

In this work, parallel prefetching in Monte Carlo molecular simulations is demonstrated in the canonical ensemble. The one-component Lennard-Jones fluid is a sufficient case study to demonstrate the efficiency of prefetching because it can be used to vary the acceptance probability and the overhead. The former is varied by the maximum displacement of the trial, and the latter is varied by potential cutoff distance. Implementation details are discussed to reduce parallel overhead in a general, modular code via caching of random numbers and potential energies. Source code is made available to the public in the prefetch plugin of the Free Energy and Advanced Sampling Simulation Toolkit (FEASST).[43]

This manuscript is organized as follows. The prefetch algorithm for Monte Carlo simulations is described in more detail in Section 2.1. The main factors influencing prefetch efficiency are discussed in Section 2.2. Benchmark simulations of a Lennard-Jones fluid and the computed efficiency metric are defined in Section 2.3. The diffusion coefficients, optimal acceptance probabilities and efficiency results are discussed in Section 3 and conclusions are provided in Section 4.

2 Computational Methods

2.1 A Prefetching Monte Carlo Algorithm for Molecular Simulation

The prefetching parallelization algorithm for Monte Carlo molecular simulations performs a number of trials in parallel and then reconstructs the Markov chain as if these trials were performed in serial. Before fully describing prefetching, a Monte Carlo trial is first described. In the canonical ensemble, a particle is selected randomly and displaced in each dimension by a random direction and magnitude up to a maximum displacement, δ. The change in energy due to the displacement is then computed to obtain the Metropolis acceptance probability for acceptance of the trial, p.[3] The average trial acceptance probability, p varies inversely with δ. Finally, the trial attempt is accepted or rejected, and ensemble averages are accumulated.

One variant of prefetching, described in Algorithm 1 and illustrated in Figure 1, is summarized as follows. At the beginning of a batch, there are n parallel threads containing identical configurations but different pseudo-random number generator seeds. The order of the trials is pre-assigned via the index of the thread, t. Each thread performs an independent, randomly generated trial attempt in parallel and waits for all other threads to complete their attempt. Any result of a call to a random number generator or potential energy is stored to enable exact imitation of this attempt without computing interactions. The accepted attempt, t_a, is defined as the minimum t of all accepted attempts. For each thread, t, delete the trial if $t > t_a$. Deleting a trial means to return the configuration to the initial state as if no trial was attempted, and is not the same as rejecting the trial. Then, for each thread, j, imitate rejections for $t < t_a$, where $j \neq t$. Imitation of rejected trials may be as simple as updating the number of attempts for trial acceptance probability, although transition-matrix Monte Carlo or other algorithms may require more information. If t_a exists, for each thread, $t \neq t_a$, imitate accepted trial t_a. To reduce overhead, a caching procedure for imitation of accepted trials with stored random numbers and potential energies is described. Finally, an

example of why ensemble averages must be computed at the end of any trial, and not only at the end of a batch, is provided.

Algorithm 1 prefetching Monte Carlo.

```
for all threads, t, in parallel do
    Create copies of self.
    Assign unique random number seeds.
end for
for all batches do
    for all threads, t, in parallel do
        Perturb configuration.
    end for
    Determine first accepted thread, t_a.
    for all threads, t > t_a, in parallel do
        Delete t.
    end for
    for all threads, t < t_a do
        for all threads, j ≠ t do
            Imitate rejected t in j.
        end for
        Ensemble average t = 0.
    end for
    if t_a exists then
        for all threads, t, in parallel do
            if t ≠ t_a then
                Imitate accepted t_a in t.
            end if
        end for
        Ensemble average t = 0.
    end if
end for
```

Each thread caches floating point numbers returned by the potential energy and random number generator objects during the initial perturbation of all threads in a batch. When the first accepted trial is determined, the other threads synchronize to the same final state by imitating the first accepted trial. This was done by unloading the floating point numbers of the cache object owned by the accepted thread in the correct order for the other threads to use as input during the trial imitation. Thus, no additional energy calculations or random numbers are required in trial imitation and synchronization. If the trials are only single-

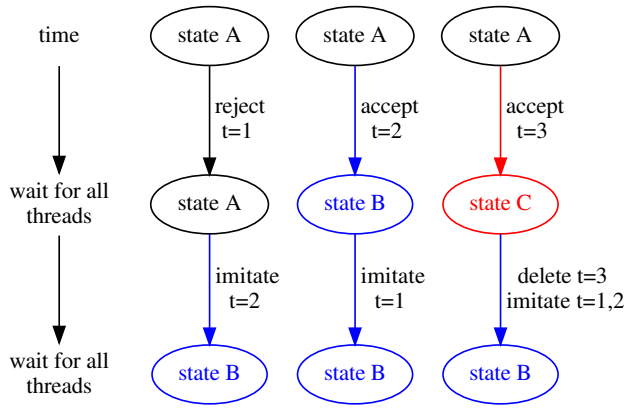

Figure 1: One example prefetching batch with $n = 3$ threads is shown. Each state represents a configuration on an individual thread. Each row of states, starting from the top, must be reached by all threads before proceeding to the next row. While the first thread, $t = 1$, is rejected, the second and third threads, $t = 2$ and 3, are accepted. Thus, $t_a = 2$ and not 3, even if $t = 3$ happens to complete before $t = 2$ in real time. The last step is to synchronize all threads as if the trials were performed in serial. Synchronization is the parallel overhead cost. In addition, deletion of $t = 3$ is wasted computation and represents a limit to the parallel efficiency for the case of negligible overhead.

particle displacements, trial imitation and synchronization refer to simply moving the same particle that was perturbed in the first accepted trial to the same location in all other threads so that all threads are in the same state to begin the next parallel batch. Thus, the following caching procedure may seem more complex than necessary in a single component canonical ensemble simulation. But the caching procedure remains the same for more complex simulations that may be considered in future work, such as configurational bias,[23] collective moves,[44] alchemical transformations[45] or particle insertion and deletion. Cache objects take other Cache objects as input to begin unloading their stored values, as demonstrated in FEASST.[43] Both the random number generator object and Potential object contain Cache objects, and they unload by input of the same type of object as themselves. Thus, unloading the Cache of one thread into another is as simple as referencing the objects of the same type from a different thread. To maintain modularity with more complex potential functions not used in this work, such as stored Ewald vectors[2] and cell-lists with dual-cut configurational bias,[38] synchronization may also be achieved efficiently by using a generalized data structure accessible to base classes to avoid frequent copying of large quantities of data in memory.

As in serial Monte Carlo simulations, ensemble averages must be computed every trial, every fixed number of trials or with equal probability for each trial, regardless if the trial was accepted or rejected. Ensemble averages should not be computed on a per batch basis. Erroneously accumulating the ensemble average only at the end of a batch would bias states to those that were recently accepted. This is because prefetch batches may have a variable number of trials, more likely ending with an accepted trial. To illustrate this point, consider one particle that may exist in one of two states with unequal potential energy at finite temperature. A Monte Carlo simulation with state change trials would erroneously compute the ensemble average energy to be the average energy of the two states, regardless of temperature, if energies were accumulated only when trials were accepted. Similarly, a prefetch simulation with very large batches, each terminated after a trial is accepted, would result in the same spurious ensemble average if only accumulated at the end of each batch. In Algorithm 1, ensemble averages are arbitrarily chosen to be computed only on thread $t = 0$ for simplicity because most ensemble average calculations are expected to use a negligible amount of CPU time. More computationally expensive ensemble averages are typically computed every fixed number of accepted or rejected trials, as was done in this work for the mean squared displacement calculation described in Section 2.3.

2.2 The Ideal Maximum Efficiency

In order to maximize the efficiency of prefetching, there are two costs to consider, as illustrated in Figure 1. The first cost is the ideal maximum efficiency that is the wasted computational time spent on attempts not incorporated into the Markov chain due to a trial accepted on a different thread. This ideal maximum efficiency approaches unity as the trial acceptance probability decreases. The second cost is the parallel overhead, which is given by both the time required for the processors to wait for the completion of the other processors as well as the time required to synchronize all of the threads back to the same state. This overhead cost may become negligible for models with expensive energy calculations and

reduces for algorithms with more efficient imitation of trials.

The ideal maximum efficiency is governed solely by p and n. To recover the serial Markov chain, one must delete trials performed on threads, t, beyond the first thread accepted, $t > t_a$. This is illustrated in the second step of the third thread in Figure 1. This deletion is required because all threads must begin a batch in the same initial state. Even rejected trials for $t > t_a$ must be deleted and cannot contribute to the ensemble averages. Note that deleted trials are distinguished from rejected trials. Deleted trials are treated as if they were never performed, while rejected trials contribute to ensemble averages and algorithms such as transition-matrix Monte Carlo. The number of deleted trials is directly related to the loss of efficiency. Assuming negligible overhead, the ideal maximum efficiency, η, is defined as the ratio of the number of trials, not including deleted trials, divided by the total number of trials, including deleted trials, and is shown in Figure 2 and given by[12]

$$\eta = \frac{1 - (1 - p)^n}{pn}. \tag{1}$$

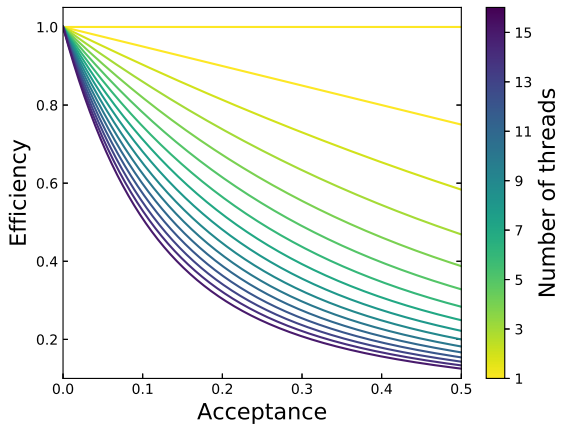

Figure 2: The ideal maximum efficiency, η, assuming negligible parallel overhead, as a function of the acceptance probability, p, and the number of threads, n.

2.3 Lennard-Jones Benchmark Simulations

A bulk, single-component Lennard-Jones liquid was chosen to benchmark the efficiency of the prefetch algorithm using OpenMP (see disclaimer in Section 5). The choice of the system is relatively inconsequential as long as the trial acceptance probability, p and the overhead can be adjusted. For these simulations, p is controlled by δ, the random uniform maximum displacement in each dimension. The relative parallel overhead is controlled by $r_c/\sigma = L/2$, the truncation distance of the Lennard-Jones 12-6 potential, where $L = 6$, 8 or 10 is the side length of a cubic periodic boundary. As used in previous studies of efficiency,[39,46] this study considered a state near the triple point with reduced number density, $\rho\sigma^3 = 0.85$, and reduced temperature, $T/\epsilon = 0.88$, where ϵ is the Lennard-Jones well depth. An integer number of particles, ρL^3 rounded down, were randomly added to the box while rejecting large overlaps. This resulted in 183, 435 or 850 particles for $L/\sigma = 6$, 8 or 10, respectively. Particle coordinates were not wrapped inside the cubic periodic boundary conditions during the simulation in order to compute the mean squared displacements. A target acceptance probability, p, was obtained by changing δ by 1% every 10^6 trials. The following pairs, $(p, \delta/\sigma)$ were approximately obtained from short simulations and used as initial conditions: (0.05, 0.35), (0.075, 0.29), (0.1, 0.26), (0.125, 0.2375), (0.15, 0.215), (0.175, 0.2), (0.2, 0.185), (0.225, 0.1725), (0.25, 0.16), (0.275, 0.15), (0.3, 0.14), (0.35, 0.125), (0.5, 0.085). Equilibration was performed for 10^7 trials and production for 10^8 trials.

Mean squared displacements were calculated every 10^5 trials and new origins were created every 10^7 trials (i.e., 10 origins).[3] The slope of the mean squared displacement, D/σ^2, was obtained by a least-squared fit up to 10^7 trials. The units for D were then converted from inverse trials to inverse CPU-hours by multiplication with the total number of trials and division by the total CPU-hours utilized by all threads. Thus, D is defined to be a real-time diffusion coefficient based on the total CPU time and not Monte Carlo trials. A higher diffusion coefficient means the particles moved further in the simulation domain for a given combined real-time CPU-hours for all threads. Perfect parallel scaling would result in the

same D as n increases, while overhead and other inefficiencies result in D decreasing with n. Thus, an appropriate measure of the efficiency in the canonical ensemble is $D(n)/D_0 \leq 1$, where D_0 is the diffusion coefficient from a serial simulation at the same L.

Benchmark simulations were run on nodes with dual-socket Intel Xeon CPU E5-2640 processors with a total of 12 cores at a clock speed of 2.5 gigahertz (see disclaimer in Section 5). Multiple simulations of the same number of threads were run on a given node with different random number generators in order to quantify uncertainty and facilitate simulations ending at nearly the same time. This limits boosted clock speeds of under-utilized nodes, which could falsely reduce the run time of the longest simulations. Thus, nodes with $n = 1$ contained 12 independent simulations, $n = 2$ contained 6, $n = 3$ contained 4, $n = 4$ contained 3, $n = 6$ contained 2 and nodes with $n = 12$ contained 1 independent simulation. For $n = 1$, both serial and prefetching simulations were benchmarked to quantify overhead. Each condition was run with 36 independent replicas, resulting in over 3 000 simulations. Error bars where obtained by multiplying the standard deviation of the mean by 2.0281 to correspond with 95% confidence.

3 Results and Discussion

In this section, prefetch scaling with number of threads is demonstrated, as well as the importance of the trial acceptance probability in determining the efficiency of the prefetch algorithm. While acceptance probabilities of $p \approx 0.225$ are optimal for serial canonical ensemble simulations, as reported previously,[39] lower acceptance probabilities become optimal as the number of parallel threads, n, increases. The maximum efficiency is approached and the effect of various parallel overhead costs are demonstrated by variation in the computational time required to compute the potential energy. Also note that many of the results depend upon the implementation details and code optimization. To improve reproducibility, the source code is available in the prefetch plugin of FEASST version 0.11.[43]

The real-time diffusion coefficient, D, is shown in Figure 3 over a range of p, n and L. This direct measure of the amount the particles move per total time spent by the CPU(s) during the simulation is a natural choice for a measure of efficiency in the canonical ensemble.[39,40] For the serial simulations (black solid lines), D reaches a maximum as a function of p, as shown previously,[39] which indicates that canonical simulations should strike an optimal balance between maximum displacement per trial, δ, and the trial acceptance, p. This value of p at which D is a maximum is defined as p_m. The single-thread prefetch simulation (red line, $n = 1$) is not the same as the traditional serial simulation (black) due to parallel overhead, although the Markov chains are exactly the same given the same pseudo-random number seed. Thus, the difference between $n = 1$ and traditional serial simulations diminish as r_c increases because the energy computation time becomes much greater than the overhead computation time. The real-time diffusion coefficient, D, decreases with increasing r_c as expected by scaling with the number of pair interactions (e.g., r_c^{-6} for $r_c = L/2$). The error estimates also decrease with increasing L because more particles lead to more mean-squared displacement samples. The optimal p_m may also depend on the overhead cost and how the trial is implemented (e.g., the relative computational cost of acceptance versus rejection of a trial).

Now that the $p-$ and $L-$dependence of the serial simulations have been discussed, consider the dependence on the number of parallel threads, n. The acceptance probability at maximum real-time diffusion, p_m, decreases with increasing n. For example, in Figure 3, $p_m \approx 0.225$ for $n = 1$ (black solid line) while $p_m \approx 0.15$ for $n = 4$ (green solid line). This is because the ideal max efficiency, η, increases with decreasing p (e.g., Eq. 1). The lower values of p reduce the probability that more than one of the n prefetched trials would be accepted, leading to less deleted trials. Thus, by increasing the maximum displacement, δ, a single trial can displace further (increasing D), and the reduction in p also increases the ideal efficiency, η, but at the cost of more rejected trials. To demonstrate that the (p, n)-dependence of η fully explains decreasing p_m as n increases, D is predicted for $n > 1$ by

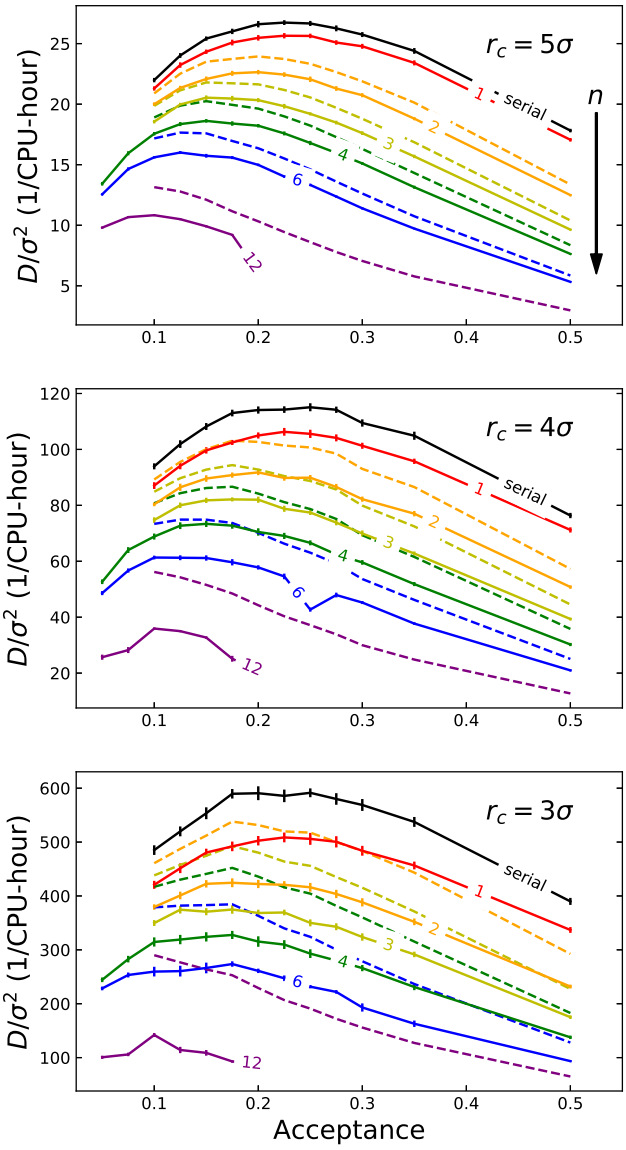

Figure 3: The diffusion coefficient, D, based upon total real-time CPU-hours is shown as a function of the trial acceptance probability, p. The line colors are labeled with the number of threads, n, ranging from a serial simulation, shown in black, to 12 threads, shown in purple. The solid lines are the computed values, while the dashed lines are the ideal efficiency predictions based on the serial simulation (e.g., multiplication of the serial D by η). Figures from top to bottom show decreasing potential cutoff distances, r_c. Error bars are for 95% confidence.

multiplying the serial D by η (e.g., dashed lines in Figure 3). When the overhead cost is relatively small (e.g., $r_c = 5\sigma$), the predictions of D and p_m using the ideal max efficiency are accurate. When the overhead cost increases (e.g, $r_c = 3\sigma$), the predictions for D are off by a proportional factor, but the predictions for p_m are still accurate. New users of this algorithm may predict the optimal p_m for a given n using serial data and the ideal maximum efficiency, η, without having to perform a series of parallel simulations. The top of Figure 4 also shows that p_m has little dependence on L within error bars, as expected.

Now that variations in p_m have been fully described, the efficiency as a function of n and r_c is shown in the bottom of Figure 4. This efficiency is defined by the ratio of the diffusion coefficient, $D(p_m)$, and D_0, the maximum D from serial simulations at the same value of L. The ratio D/D_0 is thus the measure of efficiency of prefetching as a function of n and energy computation time (varied indirectly via r_c). The prefetch simulations have $D < D_0$ due to overhead and the ideal maximum efficiency. Note that D is defined by total CPU-hours, and thus a simulation with $D/D_0 = 0.75$ for $n = 4$ will have approximately 3 times larger mean squared displacement as the serial simulation when both are run for the same elapsed time. For example, an $n = 4$ simulation run for 1 hour of elapsed time used a total of 4 CPU-hours. Thus, when the mean squared displacement, 3 times larger than the serial simulation, is divided by 4 total CPU-hours to obtain D with $n = 4$, $D/D_0 = 0.75$. In the limit of negligible overhead, the ideal maximum efficiency is shown by the solid lines which intersect the arrow in the bottom of Figure 4. The observed efficiency, D/D_0, approaches the ideal efficiency as the overhead diminishes with increased energy computation time (via L). For $r_c = 5\sigma$, a factor of 1.7 increase in speed by parallelization over 2 threads, and a factor of 3 over 4 threads, is observed. Note that a factor of 3 increase in speed over 4 threads (e.g., 0.75 efficiency) is greater than the ideal maximum efficiency, $\eta(p_m = 0.225, n = 4) = 0.71$, if the p_m from the serial simulation was used. While $p_m \approx 0.225$ for $n = 1$, $p_m \approx 0.15$ for $n = 4$. Thus, optimization of p allows for greater efficiency than holding p constant, with further improvements possible by reducing overhead.

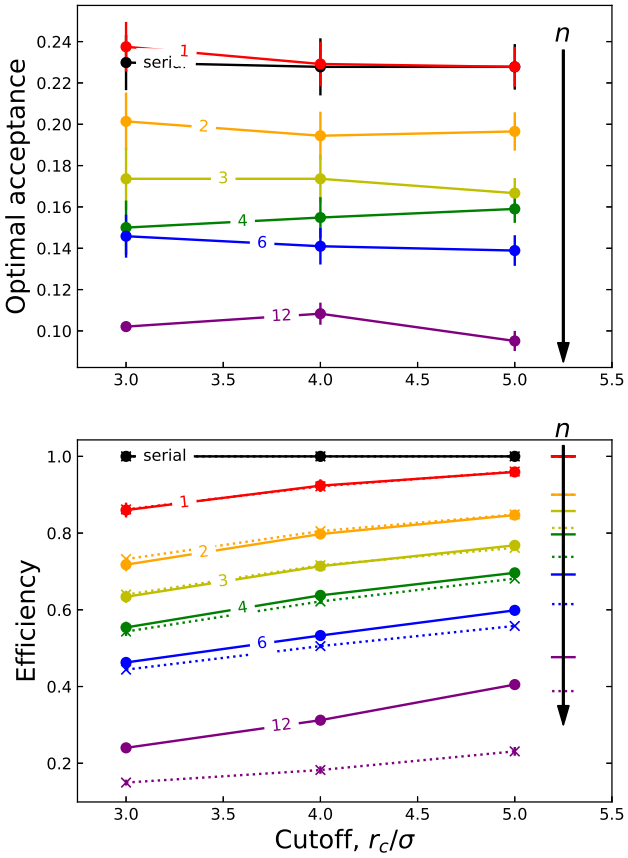

Figure 4: (Top) The optimal trial acceptance probability, p_m that results in the maximum real-time diffusion coefficient, D, as a function of the potential cutoff distance, r_c. The line colors are labeled with the number of parallel threads, n, as described in the caption of Figure 3. (Bottom) The efficiency, as measured by the ratio of the maximum real-time diffusion coefficient, D, to the serial real-time diffusion coefficient, D_0. The solid lines intersecting the arrow to the right side show the ideal maximum efficiency, η from Equation 1 at the approximate p_m. The dotted lines show the efficiency from an alternative measure based on the number of trials per second. Error bars are for 95% confidence. When applying the variance formula to obtain error bars for D/D_0, correlations between D and D_0 were neglected.

If efficiency were simply measured based on the number of Monte Carlo trials performed per CPU time, then one would obtain the dotted lines shown in the bottom of Figure 4. This alternate measure of efficiency shown by the dotted lines also does not rely upon calculation of the mean squared displacement. In this case, the acceptance probability, p_m is not optimized but rather arbitrarily selected to be $p_m = 0.2$ for all n. Failure to optimize p_m leads to the largest decreases in efficiency for the larger $n = 6$ and 12 simulations. The largest difference observed for $n = 12$ and $L/\sigma = 10$ shows nearly doubled efficiency when p_m is optimized. But for $n < 4$, the efficiency computed by this simple metric is approximately the same as that computed via mean squared displacement.

4 Conclusions

Factors of 1.7 speedup by parallelization over 2 threads and 3 speedup over 4 threads were observed using prefetching Monte Carlo simulations of a Lennard-Jones fluid in the canonical ensemble. The highest efficiencies are obtained by optimization of the trial acceptance probability as a function of the number of parallel threads, which is conveniently and accurately predicted using data from traditional serial simulations only. The single-component Lennard-Jones fluid is a sufficient benchmark system because it allows independent variation of the following two most important factors for efficiency: trial acceptance probability and parallel overhead relative to energy computation time. Although $r_c = 5$ is larger than typically used in Lennard-Jones simulations, the increase in computational expense from $r_c = 3$ to 5 affects the benchmark in a similar way to using a multi-site or charged model. The implementation of the prefetching parallel algorithm used in this work is available as open-source software via the prefetch plugin of FEASST.[43]

Many variations to the prefetch algorithm are beyond the scope of this work. If there were trials with different expected computational times, one could allow individual threads to attempt multiple fast trials while waiting. Optimizations could include automatic termi-

nation of a thread if an lower-indexed thread was already accepted. And if the expected trial CPU times are approximately known, the trials could be load balanced on each thread. More efficient implementations of Monte Carlo may also affect these results. For example, cell lists or spatial heterogeneity could affect the overhead if some threads spend more computational time than others. The prefetch efficiency with configurational bias, as well as heterogeneous trials (e.g., cluster moves, grand canonical insertions and deletions, alchemical transformations) are also of interest. In particular, parallelization of grand canonical insertion[47] and deletion is promising when their acceptance probabilities are low. These algorithmic variations are planned to be the subject of a future manuscript. While the focus of this work is upon efficiency, the number of Monte Carlo trials required to obtain the desired convergence of a property is not investigated.

5 Acknowledgements and Disclaimer

This manuscript was funded by the National Institute of Standards and Technology and is not subject to U.S. Copyright. Certain commercial firms and trade names are identified in order to specify the usage procedures adequately for reproducibility. Such identification is not intended to imply recommendation or endorsement by the National Institute of Standards and Technology, nor is it intended to imply that related products are necessarily the best available for the purpose. The author acknowledges Dr. Vincent K. Shen, Dr. Raymond D. Mountain and Dr. Joseph E. Curtis for discussions.

References

(1) Altekar, G.; Dwarkadas, S.; Huelsenbeck, J. P.; Ronquist, F. Parallel Metropolis coupled Markov chain Monte Carlo for Bayesian phylogenetic inference. *Bioinformatics* **2004**, *20*, 407–415.

(2) Allen, M. P.; Tildesley, D. J. *Computer simulation of liquids*; Clarendon Press, 1989.

(3) Frenkel, D.; Smit, B. *Understanding Molecular Simulation: From Algorithms to Applications*; Academic Press, 2002.

(4) Manousiouthakis, V. I.; Deem, M. W. Strict detailed balance is unnecessary in Monte Carlo simulation. *J. Chem. Phys.* **1999**, *110*, 2753–2756.

(5) Adams, D. J. Grand canonical ensemble Monte Carlo for a Lennard-Jones fluid. *Mol. Phys.* **1975**, *29*, 307–311.

(6) Shah, J. K.; Marin-Rimoldi, E.; Mullen, R. G.; Keene, B. P.; Khan, S.; Paluch, A. S.; Rai, N.; Romanielo, L. L.; Rosch, T. W.; Yoo, B.; *et al.* Cassandra: An open source Monte Carlo package for molecular simulation. *J. Comput. Chem.* **2017**, *38*, 1727–1739.

(7) Palmer, J. C.; Martelli, F.; Liu, Y.; Car, R.; Panagiotopoulos, A. Z.; Debenedetti, P. G. Metastable liquid-liquid transition in a molecular model of water. *Nature* **2014**, *510*, 385–388.

(8) Guo, J.; Haji-Akbari, A.; Palmer, J. C. Hybrid Monte Carlo with LAMMPS. *J. Theor. Comput. Chem.* **2018**, *17*, 1840002.

(9) Palmer, J. C.; Haji-Akbari, A.; Singh, R. S.; Martelli, F.; Car, R.; Panagiotopoulos, A. Z.; Debenedetti, P. G. Comment on "The putative liquid-liquid transition is a liquid-solid transition in atomistic models of water" [I and II: J. Chem. Phys. 135, 134503 (2011); J. Chem. Phys. 138, 214504 (2013)]. *J. Chem. Phys.* **2018**, *148*, 137101.

(10) Jones, D. M.; Goodfellow, J. M. Parallelization strategies for molecular simulation using the Monte Carlo algorithm. *J. Comput. Chem.* **1993**, *14*, 127–137.

(11) Brockwell, A. E. Parallel Markov chain Monte Carlo Simulation by Pre-Fetching. *J. Comput. Graph. Stat.* **2006**, *15*, 246–261.

(12) Byrd, J. M. R.; Jarvis, S. A.; Bhalerao, A. H. Reducing the run-time of MCMC programs by multithreading on SMP architectures. 2008 IEEE International Symposium on Parallel and Distributed Processing. Miami, FL, USA, 2008; pp 1–8.

(13) Strid, I. Efficient Parallelisation of Metropolis-Hastings Algorithms Using a Prefetching Approach. *Comput. Stat. Data Anal.* **2010**, *54*, 2814–2835.

(14) Calderhead, B. A general construction for parallelizing Metropolis-Hastings algorithms. *Proc. Natl. Acad. Sci. U. S. A.* **2014**, *111*, 17408–17413.

(15) Ren, R.; Orkoulas, G. Acceleration of Markov chain Monte Carlo simulations through sequential updating. *J. Chem. Phys.* **2006**, *124*, 064109.

(16) Ren, R.; Orkoulas, G. Parallel Markov chain Monte Carlo simulations. *J. Chem. Phys.* **2007**, *126*, 211102.

(17) O'Keeffe, C. J.; Orkoulas, G. Parallel canonical Monte Carlo simulations through sequential updating of particles. *J. Chem. Phys.* **2009**, *130*, 134109.

(18) Anderson, J. A.; Jankowski, E.; Grubb, T. L.; Engel, M.; Glotzer, S. C. Massively parallel Monte Carlo for many-particle simulations on GPUs. *J. Comput. Phys.* **2013**, *254*, 27–38.

(19) Mick, J.; Hailat, E.; Russo, V.; Rushaidat, K.; Schwiebert, L.; Potoff, J. GPU-accelerated Gibbs ensemble Monte Carlo simulations of Lennard-Jonesium. *Comput. Phys. Commun.* **2013**, *184*, 2662–2669.

(20) Hailat, E.; Russo, V.; Rushaidat, K.; Mick, J.; Schwiebert, L.; Potoff, J. Parallel Monte Carlo simulation in the canonical ensemble on the graphics processing unit. *Int. J. Parallel Emergent Distrib. Syst.* **2014**, *29*, 379–400.

(21) Rane, K. S.; Murali, S.; Errington, J. R. Monte Carlo Simulation Methods for Computing Liquid-Vapor Saturation Properties of Model Systems. *J. Chem. Theory Comput.* **2013**, *9*, 2552–2566.

(22) Witman, M.; Mahynski, N. A.; Smit, B. Flat-Histogram Monte Carlo as an Efficient Tool To Evaluate Adsorption Processes Involving Rigid and Deformable Molecules. *J. Chem. Theory Comput.* **2018**, *14*, 6149–6158.

(23) Siepmann, J. I.; Frenkel, D. Configurational bias Monte Carlo: a new sampling scheme for flexible chains. *Mol. Phys.* **1992**, *75*, 59–70.

(24) Esselink, K.; Loyens, L. D. J. C.; Smit, B. Parallel Monte Carlo simulations. *Phys. Rev. E* **1995**, *51*, 1560–1568.

(25) Vlugt, T. J. H. Efficiency of Parallel CBMC Simulations. *Mol. Simul.* **1999**, *23*, 63–78.

(26) Frenkel, D. Speed-up of Monte Carlo simulations by sampling of rejected states. *Proc. Natl. Acad. Sci. U.S.A.* **2004**, *101*, 17571–17575.

(27) Kapfer, S. C.; Krauth, W. Sampling from a polytope and hard-disk Monte Carlo. *Journal of Physics: Conference Series* **2013**, *454*, 012031, Publisher: IOP Publishing.

(28) Michel, M.; Kapfer, S. C.; Krauth, W. Generalized event-chain Monte Carlo: Constructing rejection-free global-balance algorithms from infinitesimal steps. *J Chem. Phys.* **2014**, *140*, 054116.

(29) Mattson, W.; Rice, B. M. Near-neighbor calculations using a modified cell-linked list method. *Comput. Phys. Commun.* **1999**, *119*, 135–148.

(30) Chen, Q. P.; Xue, B.; Siepmann, J. I. Using the k-d Tree Data Structure to Accelerate Monte Carlo Simulations. *J. Chem. Theory Comput.* **2017**, *13*, 1556–1565.

(31) Howard, M. P.; Statt, A.; Madutsa, F.; Truskett, T. M.; Panagiotopoulos, A. Z. Quantized bounding volume hierarchies for neighbor search in molecular simulations on graphics processing units. *Comput. Mater. Sci.* **2019**, *164*, 139–146.

(32) Panagiotopoulos, A. Z.; Wong, V.; Floriano, M. A. Phase Equilibria of Lattice Polymers from Histogram Reweighting Monte Carlo Simulations. *Macromolecules* **1998**, *31*, 912–918.

(33) Hatch, H. W.; Mittal, J.; Shen, V. K. Computational study of trimer self-assembly and fluid phase behavior. *J. Chem. Phys.* **2015**, *142*, 164901.

(34) Virnau, P.; Müller, M. Calculation of free energy through successive umbrella sampling. *J. Chem. Phys.* **2004**, *120*, 10925–10930.

(35) MacDowell, L. G.; Shen, V. K.; Errington, J. R. Nucleation and cavitation of spherical, cylindrical, and slablike droplets and bubbles in small systems. *J. Chem. Phys.* **2006**, *125*, 034705.

(36) Hatch, H. W.; Krekelberg, W. P.; Hudson, S. D.; Shen, V. K. Depletion-driven crystallization of cubic colloids sedimented on a surface. *J. Chem. Phys.* **2016**, *144*, 194902.

(37) Hatch, H. W.; Mahynski, N. A.; Murphy, R. P.; Blanco, M. A.; Shen, V. K. Monte Carlo simulation of cylinders with short-range attractions. *AIP Advances* **2018**, *8*, 095210.

(38) Vlugt, T. J. H.; Martin, M. G.; Smit, B.; Siepmann, J. I.; Krishna, R. Improving the efficiency of the configurational-bias Monte Carlo algorithm. *Mol. Phys.* **1998**, *94*, 727–733.

(39) Mountain, R. D.; Thirumalai, D. Quantative measure of efficiency of Monte Carlo simulations. *Physica A: Statistical Mechanics and its Applications* **1994**, *210*, 453–460.

(40) Kolafa, J. On optimization of Monte Carlo simulations. *Mol. Phys.* **1987**, *63*, 559–579.

(41) Jacucci, G.; Rahman, A. Comparing the efficiency of Metropolis Monte Carlo and molecular-dynamics methods for configuration space sampling. *Il Nuovo Cimento D* **1984**, *4*, 341–356.

(42) Chapman, W.; Quirke, N. Metropolis Monte Carlo simulation of fluids with multiparticle moves. *Physica B+C* **1985**, *131*, 34–40.

(43) Hatch, H. W.; Mahynski, N. A.; Shen, V. K. FEASST: Free Energy and Advanced Sampling Simulation Toolkit. *J. Res. Natl. Inst. Stan* **2018**, *123*, 123004.

(44) Liu, J.; Luijten, E. Generalized geometric cluster algorithm for fluid simulation. *Phys. Rev. E* **2005**, *71*, 066701.

(45) Hatch, H. W.; Jiao, S.; Mahynski, N. A.; Blanco, M. A.; Shen, V. K. Communication: Predicting virial coefficients and alchemical transformations by extrapolating Mayer-sampling Monte Carlo simulations. *J. Chem. Phys.* **2017**, *147*, 231102.

(46) Thirumalai, D.; Mountain, R. D. Ergodic convergence properties of supercooled liquids and glasses. *Phys. Rev. A* **1990**, *42*, 4574–4587.

(47) Daly, K. B.; Benziger, J. B.; Debenedetti, P. G.; Panagiotopoulos, A. Z. Massively parallel chemical potential calculation on graphics processing units. *Comput. Phys. Commun.* **2012**, *183*, 2054–2062.

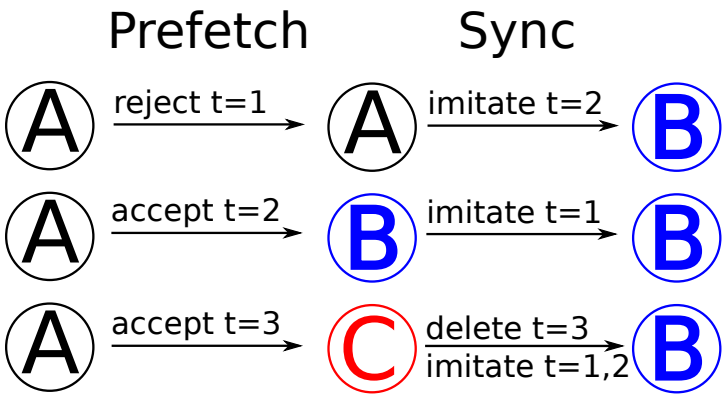

Figure 5: TOC Graphic