JCTC Journal of Chemical Theory and Computation

ARTICLE

pubs.acs.org/JCTC

# Molecular Monte Carlo Simulations Using Graphics Processing Units: To Waste Recycle or Not?

Jihan Kim,*[†] Jocelyn M. Rodgers,[‡] Manuel Athènes,[§] and Berend Smit*[∥]

[†]Lawrence Berkeley National Laboratory, [‡]Physical Biosciences Division, Lawrence Berkeley National Laboratory, Berkeley, California 94720, United States

[§]Service de Recherches de Métallurgie Physique - CEA/Saclay, 91191 Gif-sur-Yvette, France

[∥]Department of Chemical and Biomolecular Engineering and Department of Chemistry, University of California, Berkeley, California 94720, United States

**ABSTRACT:** In the waste recycling Monte Carlo (WRMC) algorithm,[1] multiple trial states may be simultaneously generated and utilized during Monte Carlo moves to improve the statistical accuracy of the simulations, suggesting that such an algorithm may be well posed for implementation in parallel on graphics processing units (GPUs). In this paper, we implement two waste recycling Monte Carlo algorithms in CUDA (Compute Unified Device Architecture) using uniformly distributed random trial states and trial states based on displacement random-walk steps, and we test the methods on a methane−zeolite MFI framework system to evaluate their utility. We discuss the specific implementation details of the waste recycling GPU algorithm and compare the methods to other parallel algorithms optimized for the framework system. We analyze the relationship between the statistical accuracy of our simulations and the CUDA block size to determine the efficient allocation of the GPU hardware resources. We make comparisons between the GPU and the serial CPU Monte Carlo implementations to assess speedup over conventional microprocessors. Finally, we apply our optimized GPU algorithms to the important problem of determining free energy landscapes, in this case for molecular motion through the zeolite LTA.

## 1. INTRODUCTION

With the introduction of multicore chips, a new paradigm of scientific computing has emerged in which scientific application codes took advantage of on-chip parallelism provided by the hardware. As computing capabilities move toward an era of exascale computing, various hardware, such as many-core CPUs, GPUs, and CPU−GPU fusion architectures are emerging to provide the next important shift in the area of high performance computing. Originally intended to handle computation for graphics, GPU scientific computing has introduced new parallelization techniques that are being utilized in solving scientific problems. Compared to CPUs, GPUs have more transistors devoted to data processing as opposed to cache memory and loop control, and accordingly programs that can be efficiently mapped onto this multithreaded hardware can see significant performance improvement. To achieve efficient computations, the GPU and CPU can work together in a heterogeneous coprocessing computing model where the sequential part of the code can be executed inside the CPU while the computationally intensive massively parallel part of the code can be accelerated inside the GPU. Traditionally, GPUs have been used mostly for graphics intensive applications but the release of NVIDIA's CUDA has allowed programmers to use C-like syntax language for code development, extending its utility for scientific computing.[2] Thus far in computational chemistry, there has been substantial GPU code development in both molecular dynamics (MD)[3] and Monte Carlo (MC) simulations.[4−6] MC simulations are very similar to MD, and as such, many of the techniques developed for molecular dynamics simulations such as neighbor lists and cell lists[7] can also be used in a Monte Carlo simulations. However, for some systems for which molecular dynamics simulations can be extremely time-consuming or simply not feasible, special Monte Carlo algorithms can make these simulations orders of magnitude more efficient.[7] Of particular interest are simulations of open systems, which rely on the addition or removal of particles. Such systems can be conveniently simulated in the grand-canonical ensemble using a Monte Carlo simulation. In this paper, we shall focus on techniques to accelerate the MC simulations of molecular systems on the GPU that expand beyond the acceleration present in typical GPU-based MD codes.

In essence, this paper seeks to address the broad question of how we may best leverage GPU resources in conducting *molecular Monte Carlo* simulations. Here, we consider three alternative avenues to accelerating convergence of a simulated thermodynamic property—the average energy per molecule. We then apply these acceleration strategies to the estimation of free energies, an important and difficult goal of molecular simulation. The three strategies are orthogonal and involve (1) an embarrassingly parallel implementation of many side-by-side Monte Carlo simulations, (2) the parallelization of the pairwise summation of Lennard-Jones (LJ) interactions, and (3) the use of multiproposal Monte Carlo coupled with waste recycling on the rejected states. The first avenue, embarrassingly parallel simulations, is certainly available to both molecular dynamics and Monte Carlo simulations; however, the small number of mobile molecules in these Monte Carlo simulations make this parallelization strategy tractable. The motivation for this approach lies in using each GPU thread to gather as much independent statistics as possible. The second avenue, parallelization of the

**Received:** July 8, 2011
**Published:** September 22, 2011

pairwise summation of LJ energies, is more pertinent to MC simulations, while in principle being amenable to MD simulations. While the LJ pair potential is employed in molecular dynamics simulations, there are two distinctions in its application in these MC simulations. For MD, all particles are moved at each time step, requiring a reevaluation of all pair forces at each time step, while in MC, one particle is moved for each MC step, resulting in fewer pairs contributing to changes in interactions at each step. Furthermore, as MC is only concerned with changes in the total energy of the system, reduction on the GPU of the sum over pair interactions results in a single systemwide energy, rather than distinct forces applied to each individual particle, resulting in a simpler reduction algorithm. These factors combine to yield a simplified parallelization scheme for LJ interactions. The goal of this approach is to conduct each single, traditional Monte Carlo step as quickly as possible. The third avenue, multiproposal MC combined with waste recycling, is also available solely to MC simulations and is an option for the use of computing power which is wholly novel. In essence, the threads of the GPU are employed to propose many possible new states at each MC step, and waste recycling is employed to harvest as much information from both the chosen state and the rejected states. Waste recycling has been employed in tandem with molecular dynamics previously through the use of multiple time slice estimators.[8] The route to parallelizing such an approach would rather focus on the energy and the force evaluations of the proposals that are generated successively by the molecular dynamics, while multiproposal MC algorithms are inherently parallelizable.

At this point, we would like to emphasize that the above techniques are generally applicable. However, as in any Monte Carlo simulation, the efficiency depends very much on the details of the system. Therefore, we illustrate the application of these techniques to a system of practical importance, methane adsorbed in the zeolites MFI (silicalite) and LTA. Zeolites are nanoporous materials important in catalysis and separations for (petro)chemical processes, and molecular simulation has proven invaluable in evaluating the thermodynamics and kinetics associated with molecular absorption and motion through zeolites.[9] For these systems, the pores can slow down the dynamics of the adsorbed molecules by orders of magnitude, making the use of Monte Carlo techniques crucial. In addition, adsorption isotherms are conveniently calculated in the grand-canonical ensemble, which requires the use of Monte Carlo simulations.

The paper is organized as follows. In section 2, we describe the molecular modeling and Monte Carlo simulation of methane adsorbed in zeolites. In section 3, we outline the mapping of the methane−zeolite system onto a GPU. In section 4, we describe in detail our GPU Monte Carlo algorithms as well as provide optimization techniques utilized in each of the different parallelization methods. In section 5, we present results of our simulations for the methane−MFI system for the different parallelization approaches, and we discuss the merits of each approach. We also apply our GPU acceleration schemes to determining the free energy profile of methane adsorbed in the LTA zeolite. Finally, in section 6, we summarize the important issues mentioned in the paper and briefly look ahead to future work.

## 2. ZEOLITE SIMULATIONS

**2.1. Molecular Models for Zeolites.** Zeolites are nanoporous crystalline materials. At present, there are over 200 different crystal structures, each with a different pore structure. The size of these pores ranges from a few angstroms to 2 nm. In this work, we study MFI (Figure 1a) and LTA (Figure 1b), which are both frequently studied materials.[10−12]

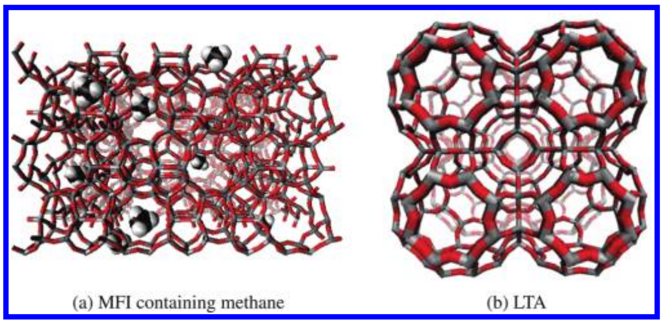


**Figure 1.** Snapshots of two zeolites with oxygen in red and silicon in gray. The adsorbate methane is shown only in MFI with carbon in black and hydrogen in white. Each zeolite is viewed from the (100) axis and contains a total of eight unit cells. The front four are opaque, and the back four are transparent. The LTA simulation box is composed of eight supercells connected to each other in the $x$, $y$, and $z$ directions via windows of smaller radius and higher free energy. The high symmetry of this structure allows us to visualize these passageways relatively easily. In the snapshot of LTA, the four opaque quandrants of the zeolite correspond to the front four supercages of the zeolite, viewed through the frontmost windows shown with thicker bonds. MFI is composed of zigzag and straight channels. The 10 methane adsorbates are most visible along the (100) axis in this instantaneous configuration at locations within the zigzag channels directed along the $x$ axis. However, the channels themselves are not visible in completion due to their kinked nature.

These two zeolites present markedly different pore topologies. MFI is composed of intersecting straight and zigzag channels. The snapshot in Figure 1a shows methane molecules adsorbed within zigzag channels, and these channels are not easily visible. In contrast, LTA is composed of an intersecting structure of large cages with narrower windows connecting them in all three directions, resulting in a simple cubic lattice of cages with free energy barriers to diffusion across each window. Figure 1b clearly displays the four frontmost cages with four posterior cages in lighter coloring.

Monte Carlo simulations have been crucial in studying these zeolite systems. In assessing molecular adsorption isotherms, Monte Carlo simulations are necessary in order to allow for particle number varying with applied chemical potential via grand canonical simulations. Furthermore, Monte Carlo simulations provide a straightforward route for calculating the free energy barriers to diffusion.[11,13] In this work, we focus on accelerating canonical MC simulations for methane in the zeolites MFI and LTA, but the lessons gleaned from this study can be broadly applicable to grand canonical simulations of this system and MC simulations of other molecular systems.

The focus of this work is on canonical (NVT) Monte Carlo simulations where the number of particles (N), volume (V), and temperature (T) remain the same throughout the simulation. Our predominant system of interest consists of methane molecules within a zeolite framework, which we assume to be rigid. The force field is parametrized using the conventional assumptions: the zeolite framework is rigid, and the interactions are dominated by the oxygen atoms in the zeolite framework. The number of methane molecules is varied for different sets of simulations while the number of framework oxygen atoms remains fixed, leading to different loadings of the framework and different effective densities of the methane molecules. In all simulations, the temperature is fixed to 300 K.

For each zeolite simulated, the simulation box is comprised of eight ($2 \times 2 \times 2$) unit cells. The three-dimensional MFI crystal unit cell has dimensions of 20.022 Å × 19.899 Å × 13.383 Å and contains 288 framework atoms. The MFI simulation box contains a total of 2304 (1536 oxygen and 768 silicon) atoms. The LTA crystal unit cell is cubic with a side length of 12.278 Å. There are 72 framework atoms inside this unit cell, resulting in a total of 576 (384 oxygen and 192 silicon) atoms in the simulation box.

Interaction between methane molecules (i.e., methane−methane) and between methane molecules and the zeolite (i.e., methane−oxygen) is modeled on a pairwise basis by the Lennard-Jones potential:

$$U(r) = 4\varepsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right] \tag{1}$$

where $r$ is the distance between two particles, $\varepsilon$ indicates the depth of the potential well (148.0 K for methane−methane and 115.0 K for methane−oxygen),[11,14] and $\sigma$ represents the effective core size of the particles with the potential well located at $2^{1/6}\sigma$. The $\sigma$ for methane−methane interactions is 3.73 Å and that for methane−oxygen interactions is 3.47 Å.[11,14] In all cases, the Lennard-Jones interaction is shifted to zero for $r > R_c$ by the subtraction of $U(R_c)$ from $U(r)$ for all $r \leq R_c = 12$ Å.

Periodic boundary conditions are imposed, and each dimension of the simulation cell is chosen to be greater than $2R_c$. This allows the determination of various properties of the molecular system through simulation of only a small subset of the entire system. As a consequence of this and through the representation of the zeolitic framework via an energy grid as discussed in section 3.2, the Monte Carlo simulations conducted in this work consist of only a small number of molecules. The largest system contains 128 methane molecules.

**2.2. Monte Carlo Simulations.** Here, we give a brief summary of the Monte Carlo techniques employed, deferring discussion of most algorithmic details until section 4. For both the embarrassingly parallel Monte Carlo simulations and the MC simulations using parallelized calculation of Lennard-Jones interactions, we simply conduct standard Metropolis Monte Carlo.[7] In this approach, an adsorbate molecule is chosen at random and moved by a random displacement chosen from $[-d_{\max},+d_{\max}]$ in each direction, generating a new state $n$ by displacement from the old state $n$. The change in energy $\Delta E$ is calculated, and the move is accepted or rejected by the Metropolis acceptance criterion:

$$\mathrm{acc}_{\mathrm{M}}(o \rightarrow n) = \min[1, \exp(-\beta\Delta E)] \tag{2}$$

We also implement two waste recycling Monte Carlo algorithms. The idea of using information of rejected moves in Monte Carlo simulations was first explored by Frenkel in his waste-recycling Monte Carlo scheme.[1] The principle idea of waste recycling is that all rejected states in a Monte Carlo simulation do carry some information, and and when included with their proper Boltzmann weights, they can improve the statistics. Delmas and Jourdain[15] have in fact proven that when MC is conducted with the symmetric Boltzmann or Barker acceptance criterion,

$$\mathrm{acc}_{\mathrm{B}}(o \rightarrow n) = \frac{\exp(-\beta E_n)}{\exp(-\beta E_o) + \exp(-\beta E_n)} \tag{3}$$

waste recycling is guaranteed to lead to faster convergence of statistical properties.

The waste-recycling method is inherently parallelizable in the sense that you may generate multiple trial states for a single Monte Carlo step, as done by Frenkel,[1] and accordingly is suited to simulate using a GPU. It is therefore interesting to investigate whether the combination of GPU with waste recycling is an attractive route for Monte Carlo simulations. The multiproposal algorithm suggested by Frenkel is most readily applied when new particle positions are chosen randomly throughout the entire simulation box, as discussed in section 4.3. However, with care, an analogue based on particle displacements may be constructed, as in section 4.4. For readers familiar with configurational bias Monte Carlo, we note that this approach for generating multiple trial states is distinct, as discussed more in Appendix .

**2.3. Simulated Properties—Energy and Free Energy.** For the bulk of our studies of GPU algorithms, we shall study the simple property of average energy of methane adsorbed in the zeolite MFI, as energies must be calculated at each step in the Monte Carlo simulation. Average energy is not typically a difficult quantity to converge; however, it yields a well-defined benchmark for gauging the speedup of our various GPU algorithms and to study the optimization of these algorithms on the GPU architecture.

Once we have optimized the various algorithms, we then apply the lessons gleaned to a highly relevant property, the free energy profile of methane diffusing through the zeolite LTA. In zeolites, these free-energy barriers are relevant for characterizing the diffusive behavior of adsorbates within the zeolite framework.[16] The zeolite LTA poses a straightforward pore topology of a simple cubic arrangement of pores separated by windows. Thus, in any axial direction, adsorbed methane molecules encounter free energetic barriers at the windows connecting the cages. This barrier is substantial yet still thermally accessible. As such, the calculation of the free-energy profile along the reaction coordinate $z$ is simply calculating the histogram of probabilities along $z$, $P(z)$:

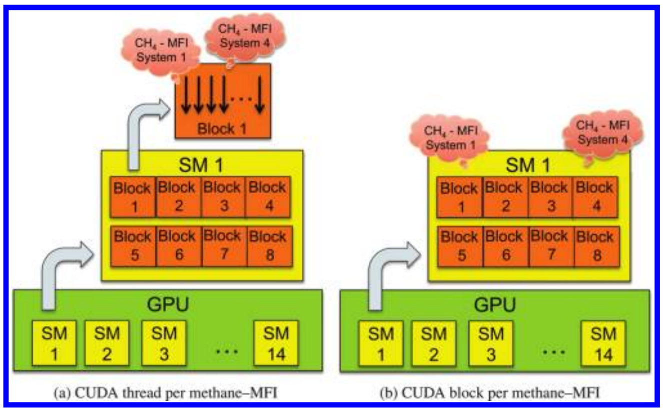
$$F(z) = -k_{\mathrm{B}}T \ln P(z) \tag{4}$$

Certainly, this examination is relevant because $F(z)$ is a meaningful and computationally intensive quantity to calculate. But this final comparison is also important to conduct because waste-recycling Monte Carlo is most fruitfully employed on CPUs for free energy calculations,[1,8,17,18] so we must also allow for this possibility on the GPU as well.

## 3. COMPUTATIONAL SETUP

In this section, we first outline the various strategies we employ for mapping these zeolite simulations onto the GPU. These strategies as well as the implementation, optimization, and testing of GPU algorithms for these strategies are the main contributions of this paper, as detailed in subsequent sections. We then describe in section 3.2 one constant in all of our studied GPU implementations—the construction of a framework energy grid to represent the interactions between methane molecules and the zeolite framework atoms. This grid in general yields a substantial speedup in the simulation of zeolites, which are rigid to a good approximation; thus careful implementation on the GPU is important.

All of the simulations for this work were conducted using the Dirac cluster at the National Energy Research Scientific Computing Center (NERSC). The CURAND library was employed for generating the necessary random numbers on the GPU.[19]



(a) CUDA thread per methane–MFI (b) CUDA block per methane–MFI

**Figure 2.** Mapping of methane−MFI systems to CUDA hardware resources. In Tesla C2050 GPU cards, there are 14 streaming multiprocessors (SM) shown as green blocks and at maximum eight resident blocks (Bl) per SM (shown as red boxes). Each thread within a block is indicated by a black arrow. (a) The CUDA thread per methane−MFI implementation mapping is used in the embarrassingly parallel algorithm (PMC). (b) The CUDA block per methane−MFI implementation mapping is used in the parallel Lennard-Jones algorithm (PLJ) and the two multiproposal waste recycling algorithms (MUP and MDP). As an example, we show the “location” of methane−MFI system 1 and system 4 on hardware resources for each mapping.

Further hardware and compilation details may be found in Appendix .

**3.1. Strategies for Mapping Zeolite Simulations onto GPU.** Given that only a small subset of the entire structure needs to be simulated, thousands of methane−MFI systems can simultaneously fit inside the GPU DRAM and multiple, independent Monte Carlo simulations can be processed in parallel to improve the statistical accuracy. We pursue two different ways to map the individual methane−MFI system to the GPU hardware.

First, we choose a strategy where each CUDA thread executes its own independent MC simulation (Figure 2a). Denoting the CUDA block size as $n_{\text{threads}}$, a single Monte Carlo simulation contains $N_{\text{blocks}} \times n_{\text{threads}}$ independent methane−MFI systems. Effectively, this algorithm can process $n_{\text{threads}}$ times more methane−MFI systems in one simulation compared to the next parallelization scheme conducted on a per block basis. Despite this benefit, this implementation includes overall longer computational wall time and more DRAM usage that reduces the maximum number of particles that can be simulated, and it also cannot utilize fast GPU memory due to the large amount of resources used by a single CUDA thread.

Therefore, we also choose an alternate parallelization strategy where each CUDA thread block executes its own independent MC simulation with threads from the same block contributing to either waste recycling or parallel Lennard-Jones calculations (Figure 2b). $N_{\text{blocks}}$ is used to represent the total number of CUDA thread blocks launched and, in this context, refers to the total number of independent methane−MFI systems processed in a single simulation. Communications between different CUDA thread blocks is unnecessary until the end of the simulation when energy values are collected from each of the blocks to obtain an ensemble average.

**3.2. Construction of the Framework Energy Grid.** In order to save computation time, explicit Lennard-Jones pair potential calculation between the methane and the framework molecules is avoided. Instead, we construct a three-dimensional rectangular grid superimposed on top of the entire simulation box, where each of the points represents the Lennard-Jones pair potential values between a single methane molecule and the entire molecular framework. All of the energy grid point values are computed just once before the start of the Monte Carlo simulation and subsequently used during the Monte Carlo simulation as a lookup table. By utilizing thousands of lightweight threads available in the GPU architecture, the energy grid point values can be computed in parallel inside the GPU using a simple domain decomposition method in which each CUDA thread is responsible for computing a single grid point value. To improve accuracy within this approximation, it is better to generate as many points as possible while working within the memory available in the GPU DRAM. With an energy grid of mesh size 512 × 512 × 256 along the $x$, $y$, and $z$ directions, results within 0.05% of the energy values from utilizing direct Lennard-Jones calculations are obtained. Therefore, we use the corresponding values of $\delta x = 0.0784$ Å, $\delta y = 0.0778$ Å, and $\delta z = 0.105$ Å as our mesh size for all of the simulations.

The memory needed to store the energy grid values is too large (roughly 500 MB for 64-bit doubles) to fit into any of the fast GPU memory, and hence the energy grid array is put into the slow, global GPU DRAM. Given that the methane molecules are free to occupy spatial coordinates not directly located on the energy grid points, linear interpolation functions from eight nearest neighboring energy grid point values are used to approximate the exact Lennard-Jones pair potential value at a given particle position at each Monte Carlo step. Because these eight neighboring points cannot be stored contiguously inside the GPU memory for all of the grid points, multiple memory transactions are needed to compute the contributions of the framework molecules from a single particle position. Although these memory transactions are expensive, there still exists considerable speedup utilizing the energy grid as opposed to explicitly computing the Lennard-Jones pair potential terms from each of the 1536 framework molecules for all Monte Carlo steps. Moreover, as the number of molecules increases, the proportional wall time spent in the energy grid read access becomes smaller and becomes less of a concern. For the GPU architecture in general, frequent data movement from the GPU DRAM causes the code to become memory bound (which amounts to 144 GB/s in the Tesla C2050 cards used in our work).

Finally, the interaction between the framework molecules (i.e., oxygen−oxygen) is ignored, as oxygen is assumed to be stationary throughout the simulation.

## 4. GPU MONTE CARLO ALGORITHMS

In this section, we describe in detail the four GPU Monte Carlo algorithms implemented in our work. In all of the methods, one initial methane−MFI system is generated by assigning randomized positions to methane molecules inside the simulation box. This configuration is equilibrated via a serialized Markov-Chain Monte Carlo (MCMC) method using a single core CPU. In this MCMC simulation, a small (that is, small compared to the dimensions of the simulation box), random translation step size is used to propose a particle movement to a new position, and this proposal is either accepted or rejected according to the Metropolis probability.[20] The step size in particle displacement is chosen such that approximately 50% of the translation proposals are accepted according to the Metropolis probability. In this work, we do not focus on accelerating the equilibration phase of the Monte Carlo, and therefore the CPU rather than the GPU is utilized to equilibrate the system. After equilibration, the particle coordinates are duplicated $N_{\text{blocks}}$ times (in the embarrassingly parallel algorithm, $N_{\text{blocks}} \times n_{\text{threads}}$) in the CPU, and the data are transferred to the GPU DRAM. In general, we want $N_{\text{blocks}}$ to be a number that is an integer multiple of the total number of streaming multiprocessors found in the GPU (e.g., 14 in the case of Tesla C2050) to balance the workload among the multiprocessors. Inside the GPU, these systems need to be decorrelated in order to remove any correlation that might persist between particles from different systems, which can adversely affect the MC results. The algorithm used for decorrelation is the same as that employed in the accumulation phase. This algorithm is unique to each method, and all are detailed next.

**4.1. Embarrassingly Parallel Monte Carlo (PMC).** In contrast to the following three methods, in the embarrassingly parallel Monte Carlo (PMC) algorithm, each CUDA thread (instead of each CUDA block) is mapped to one methane−MFI system, and all of the threads conduct their own independent monoproposal MCMC simulation. Within this implementation, since each CUDA thread needs to have its own unique data of particle positions as well as other hardware resources, the fast memory available from the GPU hardware is insufficient, and all of the data are kept inside the global DRAM. We limit global DRAM transactions via memory coalescing and utilize the following indexing scheme to store particle coordinates inside our arrays. For a given CUDA thread $j$, the coordinates of the particles are stored inside an array in indices $j$, $j + n_{\text{threads}}N_{\text{blocks}}$, $j + 2n_{\text{threads}}N_{\text{blocks}}$, etc. such that a single memory transaction executed by a *warp* (corresponding to 32 independent methane−MFI system) can load a contiguous block of 32 64-bit data related to one particle index in order to maximize memory throughput. Within this implementation, all of the CUDA threads (and therefore, each independent methane−MFI system) choose the same particle index number for displacement in a Monte Carlo step to avoid warp divergence. Once the systems are completely decorrelated from one another, choosing the same index does not cause a problem since particles that possess the same index number from different systems are unrelated to one another.

**4.2. Parallel Lennard-Jones (PLJ).** For most system sizes, the bottleneck routine in our Monte Carlo simulation is the kernel that computes the Lennard-Jones pair potential. Accordingly, we devise a GPU algorithm that parallelizes this calculation in the monoproposal MCMC algorithm. The change in total energy for moving particle $k$ from $\mathbf{r}_{\text{k,old}}$ to $\mathbf{r}_{\text{k,new}}$ while holding the remaining particles fixed can be calculated in $\mathcal{O}(N_{\text{tot}})$ time via

$$\Delta E_{\text{tot}} = \sum_{j\neq k}^{N_{\text{tot}}} U(r_{\text{jk, new}}) - \sum_{j\neq k}^{N_{\text{tot}}} U(r_{\text{jk, old}}) + E_{\text{grid}}(\mathbf{r}_{\text{k, new}}) - E_{\text{grid}}(\mathbf{r}_{\text{k, old}}) \qquad (5)$$

where $N_{\text{tot}}$ is the total number of particles in the system and $U(r_{\text{jk}})$ is the Lennard-Jones pair potential between particles $j$ and $k$ as defined in eq 1. $E_{\text{grid}}(\mathbf{r}_{\text{k}})$ represents the total summation of pair potentials between particle $k$ and all of the framework molecules as computed by an energy grid. The computation of these energies requires only linear interpolation from given grid points, as described in the previous section. The most expensive computation is the calculation of all LJ pair energies, and for a given Lennard-Jones pair-potential calculation between two particles, the most expensive operation is the floating-point division operator. In order to reduce the cost, only one division operator is executed per pair potential, and the intermediate term is reused to avoid the second division operation in eq 1.

In the PLJ algorithm, the CUDA threads within the same block divide up the computation work of the two summation terms found in eq 5. Energy subtotals from each of the CUDA threads are combined at each MC step using a reduction kernel to obtain the total Lennard-Jones potential value. We utilize a reduction kernel similar to one found in the CUDA SDK example.[21]

**4.3. Waste-Recycling with Multiple Uniform Proposals (MUP).** The previous two implementations are based on the conventional Monte Carlo algorithm and in which we use the GPU to speed up the bottleneck routines in the computation. In this section, we develop an alternative approach in which we implement the idea of using information of rejected moves in Monte Carlo simulations, using the GPU to generate multiple trial states and employing a multiproposal version of the waste recycling Monte Carlo described in section 2.2. In practice, the waste recycling algorithm can be easily mapped into conventional CPU architectures, but in the case of single-core architectures, the parallelism would be lost as each of the multiple proposed trial states in the Monte Carlo algorithm would need to be processed sequentially. On multicore CPU architectures, the different trial states can be processed in parallel, but the performance gain will not be as great compared to the execution via GPU architecture due to the larger overhead cost of generating and combining multiple CPU threads.

In our CUDA waste recycling Monte Carlo code, we first employ a variant of the waste-recycling algorithm outlined in the paper by Frenkel,[1] which we shall refer to as multiple uniform proposals (MUP). In this paper, he describes a symmetric Boltzmann-like multiproposal scheme where the original state $o$ is included on equal footing with all of the states in the set $\{n\}$ of trial states, and the final state is chosen from the set $\{o,\{n\}\}$.

Our CUDA waste recycling algorithm based on that of Frenkel[1] is as follows:

1 Generate an initial state $o$, and set to zero the accumulator $S_{\text{A}}$ for estimating the average of the observable $A$.

2 Generate a set of trial states, $\{n\}$, by randomly choosing one methane molecule and randomly generating a set of new positions for this molecule uniformly throughout the entire simulation box. The total number of trial states including the old state $o$ is represented by $N_{\text{prop}}$.

3 Compute the Boltzmann weights $w_i = \exp(-\beta E_i)$ for all $i \in \{o,\{n\}\}$.

4 Update the accumulator $S_{\text{A}}$ according to the following equation:

$$S_{\text{A}} \rightarrow S_{\text{A}} + \frac{\sum_i w_i A_i}{\sum_i w_i}$$

where again $i \in \{o,\{n\}\}$.

5 Choose a final state $f \in \{o,\{n\}\}$ with an acceptance probability

$$p_{\text{acc}}(f) = \frac{w_f}{\sum_i w_i}$$

6 Repeat steps 2−5 starting from the chosen state $f$. Continue for a total of $M$ Monte Carlo steps.

7 Estimate the average of $A$ as $\langle A \rangle_{\rm est} = S_{\rm A}/M$.

For this work, the accumulator term $S_{\rm A}$ defined in step 1 either refers to the total energy of the system or the occupation probability of a given volume slice. These individual probabilities for volume slices that span the simulation box are then combined in the end to yield $P(z)$ for free energy calculations.

For step 2, the total number of trial states, $N_{\rm prop}$, is set to be a multiple of 32 because the NVIDIA GPU hardware schedules and executes threads in groups of 32 called a *warp*. For any other number, some threads will remain idle while waiting for other threads within the same warp to finish work, resulting in warp divergence and performance degradation. We note that step 2 is the point of main difference with our alternative multiproposal waste recycling algorithm described in the following subsection.

The total energy of proposal $i$, $E_i$, is used to determine the associated Boltzmann weight, $w_i$, in step 3 and is calculated from eq 5. All $N_{\rm prop}$ threads share the same value for the second summation term in eq 5, $\sum_{j \neq k}^{N_{\rm tot}} U(r_{\rm jk,old})$, and accordingly this term needs to be only calculated once for all of the multiple proposals. In our implementation, thread 0 calculates this term. This offers an advantage over a monoproposal Lennard-Jones algorithm in which the term needs to be calculated for each new displacement, and its value cannot be reused for additional displacements.

In step 4, the accumulator term $S_{\rm A}$ is updated at each Monte Carlo step by taking the summation of the energy contributions from $N_{\rm prop}$ CUDA threads. Two separate reduction operations need to be performed to compute $\sum_i w_i A_i$ and $\sum_i w_i$, and we utilize an algorithm similar to one found in the reduction for parallel Lennard-Jones calculations. It is noteworthy to point out that array elements with nonzero indices contain partial summation results that are later used in step 5 to expedite calculation in its routine. As one example, $w[1] = \sum_{i=0}^{N{\rm prop}/2} w[2i + 1]$. Because multiple accesses to the same memory address occur as many as log $N_{\rm prop}$ times in the reduction algorithm, all of the array elements $w_i A_i$ and $w_i$ are fetched once from the global memory and moved into the fast, shared memory in order to reduce global memory bandwidth. Using shared memory in the reduction kernel is more important in pre-Fermi cards, which does not have the L1 cache memory. In the Fermi GPUs, the array can be kept inside the global memory and moved into the 16/48 kB L1 cache, which is the same hardware as the shared memory, avoiding performance degradation.

In step 5, the WRMC algorithm updates the particle coordinates by selecting a final state among all proposed trial states with probability proportional to its Boltzmann weight. This acceptance probability is a multiproposal extension of the symmetric Barker acceptance ratio defined in section 2.2.[1,15] For thread 0, the final state is set to be equal to the initial state, thereby making it possible for the system to remain unchanged after a WRMC step. Using CUDA, step 5 can be conducted in log $N_{\rm prop}$ steps by reusing intermediate results from the step 4 reduction kernel in a following way. At the end of the reduction kernel, the array element $w[0]$ contains the sum of all of the Boltzmann weights $S = \sum_i^{N_{\rm prop}} w_i$, whereas other array elements have partial sums that are less than $S$. In step 5, all of the elements in this array are divided by $S$ for normalization purposes, such that $w[0] = 1$ and $0 < w[i] < 1$ for all $i \neq 0$. Next, a random number $R$ uniformly distributed from 0 and 1 is generated per system using the curand_uniform_double function from the device API. Depending on $R$, the final state is chosen according to the algorithm outlined in the CUDA code below:

Listing 1. Step 5 CUDA Code

```
//R - random number uniformly choosen from [0,1]
//w - array that contains normalized Boltzmann weights
//step_num - index of trial state chosen for next MC move
//N - total number of independent WRMC proposals

int index = 1;
int N = blockDim.x;

for(int i = 2; i < N; i*=2)
{
if (R < w[index])
index +=i;
else
{
R -= w[index];
index += 0.5*i
}
}

//last step
if (R ≥ w[step_num])
step_num -= 0.5*N;
```

The system updates to a new state, and the waste recycling Monte Carlo step is repeated $M$ steps (step 6). Finally, the average quantity $A$ for the system is obtained after $M$ moves according to step 7.

### 4.4. Waste-Recycling with Multiple Displacement Proposals (MDP).

The waste-recycling algorithm presented in the preceding subsection relies on nonlocal moves. Such a scheme works well if the rejected configurations have a reasonable contribution. In dense systems, however, the probability that a nonlocal move gives a significant contribution is extremely low. Therefore, we expect that local moves constructed on the basis of *displacements* from the old state, as done for our PMC, PLJ, and benchmark serial CPU algorithms, will be much more fruitful. This, however, requires a modification of the original algorithm in order to construct a waste-recycling algorithm based on multiple particle displacements from the old state. As will be demonstrated, this method is more effective at exploring important regions of phase space at high particle densities.

The only needed modification of the previous algorithm lies in constructing the set of proposals $P \equiv \{o,\{n\}\}$ in step 2, where small displacement steps are used to generate the multiple trial states in our algorithm. In generating this set $P$ of positions based on displacements, it is *crucial* that the set is equally likely to be generated by any trial position in the set. Provided that the generation probability $p_{\rm gen}(P|i)$ of set $P$ is equal for any point $i \in P$, the simple Barker-like acceptance probability employed in section 4.3 may still be used.

The crux of generating such a set of positons $P$ lies in constructing two separate random walks starting from the chosen particle $k$ in state $o$. The total length of the two random walks is $N_{\rm prop} - 1$ displacements, leading to a total of $N_{\rm prop} - 1$ trial states. However, by choosing the position of the generating point from state $o$ uniformly within the combined length of the two random walks, any trial position within the random walk is *equally likely* to have generated the set of positions $P$. Therefore $p_{\text{gen}}(P|i)$ is equal for all trial states in the set, yielding the simple acceptance probability given in section 4.3 by detailed balance.

In detail, each random walk is initiated from the position of particle $k$ in state $o$. The two walks are of length $N_{\text{prop}} - l - 1$ and $l$, with $l$ representing a random integer from 0 to $N_{\text{prop}} - 1$. Each point in each random walk is based on a uniformly generated random displacement from the previous position in the random walk. We make sure that each of these random displacements abide by the periodic boundary condition given that they can fall outside of our simulation volume. We emphasize the exact algorithm we use for trial state generation by displacements because several seemingly reasonable alternatives for generating multiple proposals either result in incorrect sampling of states or a prohibitively expensive use of waste, as is discussed in Appendix

**4.5. Optimal Estimation for Waste Recycling.** The work of Delmas and Jourdain[15] suggests an optimal reweighting of waste recycling averages and traditional MC averages when employing symmetric acceptance ratios such as those employed in sections 4.3 and 4.4. To our knowledge, the work presented here is the first implementation of this optimal estimator in a multiproposal framework, though this estimator has been explored in detail in a monoproposal setting.[22]

For a quantity $A$, detailed mathematical analysis by Delmas and Jourdain demonstrated that the optimal estimation of its average, when using a Barker-like acceptance ratio, may be written as

$$\langle A \rangle_{\text{opt}} = (1 - b^*)\langle A \rangle_{\text{MC}} + b^* \langle A \rangle_{\text{WRMC}} \tag{6}$$

where optimal estimation is in the sense of minimal standard deviation. The coefficient $b^*$ itself depends on the variance of the property as well as the correlation in a property across steps in the Markov chain as

$$b^* = \frac{\langle A^2 \rangle - \langle A \rangle^2}{\frac{1}{2}\langle (A_{m+1} - A_m)^2 \rangle} \tag{7}$$

with the $A_m$ and $A_{m+1}$ in the denominator indicating quantity values at successive states in the Markov chain.[15,22] This estimator of Delmas and Jourdain is still valid when using Metropolis-like acceptance ratios such as the one described for configurational-bias MC in Appendix ; however, in such cases, the estimator is not optimal.

Since each quantity in the above equation for $b^*$ is a pure ensemble average, these averages may be evaluated with equal validity as waste-recycling estimates and as traditional Monte Carlo averages. Thus, when waste-recycling provides a benefit, which we expect for this multiproposal scenario, a waste-recycling estimation of $b^*$ makes sense. In such a case, the coefficient for the optimal estimator $b^*$ itself may be estimated as

$$b^*_{\text{WRMC}} \approx \frac{\frac{1}{M}\sum_{m=1}^{M}\sum_{i \in \{o,\{n\}\}} p_{\text{acc}}(i)A_i^2 - \left(\frac{1}{M}\sum_{m=1}^{M}\sum_{i \in \{o,\{n\}\}} p_{\text{acc}}(i)A_i\right)^2}{\frac{1}{2M}\sum_{m=1}^{M}\sum_{i \in \{o,\{n\}\}} p_{\text{acc}}(i)(A_i - A_o)^2} \tag{8}$$

Note that the denominator in the above equation no longer references the $m$th and the $(m + 1)$th states. All quantities $A_i$ are those of the proposed moves from state $o$ in a given step $m$. Thus, rather than only considering the final accepted state used in the subsequent step $m + 1$, we instead include information on all proposed paths out of state $o$ at step $m$ in both the numerator and the denominator. All summands in the numerator and denominator implicitly depend on the current step $m$ in the Markov chain; this dependence is omitted for simplification of notation.

This optimal estimator may be employed for both waste recycling algorithms studied in this paper, MUP and MDP. However, as the results based on multiple displacment proposals (MDP) are much more promising across a range of loadings, we restrict our application of the optimal estimator to simply the MDP algorithm. In these instances, we shall annotate the results as MDP-DJ, for Delmas and Jourdain.

## 5. RESULTS

The details of the GPU Monte Carlo simulation results are given in this section. The statistical accuracies for different GPU algorithms are compared for various numbers of methane molecules in both unoptimized and optimized CUDA configurations. In the unoptimized CUDA configuration, the number of blocks per SM is set to be 1, and the number of threads per block is set to be 32, resulting in suboptimal work allocation to the GPU threads. In the optimized CUDA configuration, both the number of blocks and the number of threads per block are increased to maximize occupancy in the SM and subsequently performance. Despite its inefficiency, the unoptimized CUDA configuration serves as a good basic starting configuration that allows us to easily test the correctness of the code and to compare the different GPU parallelization methods. Within the optimized CUDA configuration, we compare the performance difference for different distributions of CUDA blocks/threads for the waste recycling Monte Carlo algorithm. We analyze the effect of including the optimal estimator in the statistical accuracy of these WRMC simulations. As a performance comparison, we show speedup numbers of our GPU algorithms compared to a single core CPU code as evidence of performance benefits of using CUDA. Finally, we investigate the performance utility of the waste recycling Monte Carlo algorithm in computing the free-energy profile of methane gas molecules inside a zeolite framework.

**5.1. Initial Unoptimized Assessment.** Figure 3a displays the single iteration wall time for the embarrassingly parallel Monte Carlo (PMC), monoproposal parallel Lennard-Jones calculation (PLJ), multiple-uniform-proposals waste recycling Monte Carlo (MUP), and multiple-displacement-proposals waste recycling Monte Carlo (MDP), in that order from left to right for $N$ = 32, 48, 64, 80, 96, 112, and 128 methane molecules. The iteration time for the embarrassingly parallel Monte Carlo method is the longest for all system sizes, which is reasonable given that the number of floating point operations is the largest here since each CUDA thread conducts its own Monte Carlo simulation. The iteration time for the parallel Lennard-Jones calculation is the shortest, as $n_{\text{threads}}$ threads share the same methane−MFI system, in contrast to PLJ, and as only one proposal move per CUDA block is required at each iteration as opposed to multiple trial moves per iteration for the two waste recycling methods. In the limiting case of a very large system where Lennard-Jones calculation completely dominates total wall time, the number of floating point operations in the two waste recycling methods would be roughly $n_{\text{threads}}$ times larger than the number found in the parallel Lennard-Jones method for the same number of accumulation steps. However, this workload can be reduced by

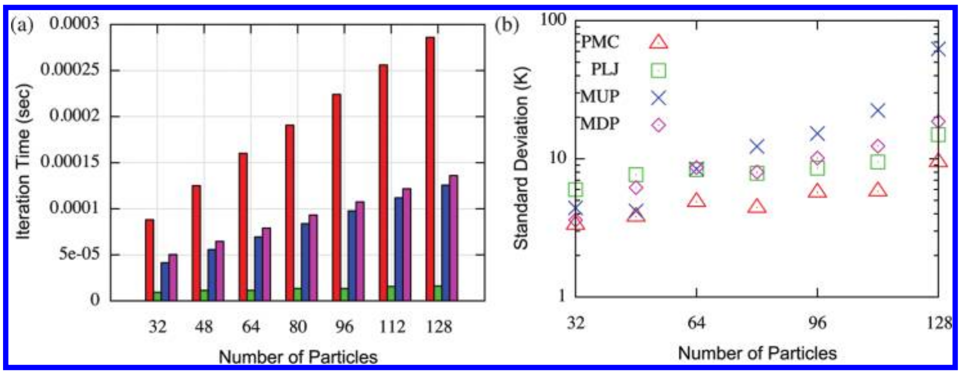


**Figure 3.** Comparison of the various parallel algorithms for different numbers of methane molecules. (a) Single MC iteration time (in seconds on the left axis) as a function of the number of methane molecules (from left to right: PMC, PLJ, MUP, MDP). (b) Comparison of the standard deviations in the energy (in units of Kelvin) as a function of the number of methane molecules for the various parallelization strategies. The number of accumulation steps is varied to equate the total wall times of the simulations in all four methods.

approximately half in the waste recycling methods as $n_{\text{threads}}$ independent proposals all share the same old system energy value at the start of each Monte Carlo step, and thus this value needs to be just calculated once for all of the multiple proposals. Overall, the wall time for the parallel Lennard-Jones method in the limiting case would be approximately $0.5n_{\text{threads}} = 16$ times faster than the waste recycling methods with the WRMC algorithms collecting 32× more energy samples during the simulation. Analyzing the wall times for the two waste recycling methods, we observe that the displacement WRMC has about a 10 $\mu$s longer iteration time than the uniformly sampled WRMC for all system sizes due to the additional overhead from generating two separate random walks in the displacement WRMC method.

As an initial comparison of the effectiveness of each technique in variance reduction, we run simulations with comparable total wall times as follows. The total number of equilibration and decorrelation steps is set to be 200 000 each while the baseline accumulation step is set to be 1 million for the embarrassingly parallel Monte Carlo method. The accumulation steps for other methods are adjusted with respect to this baseline number such that the total wall time spent inside the GPU for all of the methods remains equal. The number of accumulation steps for parallel Lennard-Jones is set to be 9.5, 11.0, 14.0, 16.7, 16.4, and 17.8 million; the number for uniformly sampled WRMC is set to be 2.1, 2.2, 2.3, 2.3, 2.3, 2.3, and 2.3 million steps, and the number for displacement WRMC is set to be 1.8, 1.9, 2.0, 2.0, 2.0, 2.1, and 2.1 million steps for $N = 32$, 48, 64, 80, 96, 112, and 128 methane molecules. The total number of CUDA blocks, $N_{\text{blocks}}$ is equal to 14 with one block occupying each of the streaming multiprocessors. The CUDA block size $n_{\text{threads}}$ is set to 32, which leads to a low occupancy number (i.e., 32/1536 = 0.021, with 1536 being the maximum number of resident threads per multiprocessor in Tesla C2050) and underutilization of the Fermi GPU hardware. Because the main focus here is to investigate the general behavior trends for each of the different CUDA parallelization methods at varying particle densities, we initially explore a simple, reduced layout of thread blocks and sizes and forego optimization analysis until later.

In Figure 3b, we plot on a log−linear scale the standard deviation of total energy for 40 independent Monte Carlo simulation runs as a function of the number of methane molecules for the four methods. Overall, the embarrassingly parallel Monte Carlo (PMC) algorithm provides the best statistical accuracy for all system sizes, as the benefit of processing $n_{\text{threads}}$ times more independent systems than other methods outweighs the cost of the longer single iteration time. In the parallel Lennard-Jones (PLJ) method, the proportional wall time spent in routines other than the Lennard-Jones kernel remains large compared to other methods (due to short single iteration wall time), and this overhead causes performance degradation that is evident especially in the low-density regime where the method fairs the worst. The multiple-uniform-proposals (MUP) WRMC excels for small system sizes/low densities where the likelihood of sampling non-negligible Boltzmann weights (i.e., low energy configurations) in the proposed trial states remains high. However, at large system sizes/high densities, the method fairs the worst, as most of the randomly generated trial states result in high energy configurations that add negligible contribution to the Monte Carlo statistics. This problem is remedied in the multiple-displacement-proposals (MDP) WRMC method where the random walks are conducted in small step sizes and lead to relatively lower energy configurations for each of the proposed trial states. From Figure 3b, we observe that the standard deviation values from the MDP waste-recycling algorithm are comparable to the ones from the PLJ method for all system sizes.

**5.2. Exploration of MDP Parameters.** Next, we further analyze the displacement WRMC method (MDP) and vary the maximum displacement step size, $d_{\max}$, as well as the total length of the random walks (i.e., $n_{\text{threads}}$, which is the CUDA thread block size) in our MC simulations to determine the optimal parameter settings for the WRMC simulation. Because our interest lies mostly in code optimization for high density systems, we restrict our analysis to the displacement WRMC method for 128 methane molecules, which provides better statistical accuracy at high density compared to the multiple-uniform-proposals WRMC. Figure 4 shows the plot of the standard deviation of the total system energy for 40 independent MC simulations as a function of $d_{\max}$ with 1 million accumulation steps for total length of random walks $n_{\text{threads}} = 32$, 64, 128, 256, and 512. In all of the simulations, $d_{\max}$ is the same value along all three spatial directions, and $n_{\text{threads}}$ is set to be a multiple of the warp size in all cases to avoid warp divergence and for simplicity. The total GPU wall time for different $n_{\text{threads}}$ are not set to be equal here and accordingly, simulations with larger $n_{\text{threads}}$ have longer single iteration times and are expected to provide greater statistical accuracies. Given the GPU hardware limit on register size per streaming multiprocessor (32 kB/SM), the maximum number of threads that can occupy a multiprocessor is 512 in our

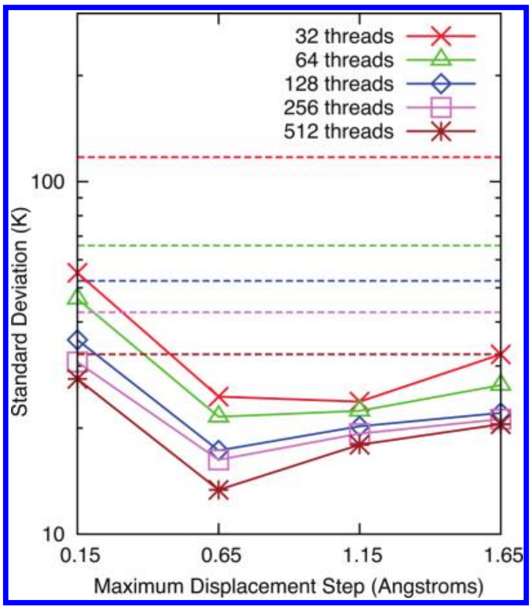


**Figure 4.** Standard deviation of the average of energy as a function of the maximum individual displacement step, $d_{max}$, for CUDA block size ranging from 32 to 512 threads. The CUDA block size corresponds to the total length of the two random walks in the multiple-displacement-proposals (MDP) WRMC algorithm. The standard deviation values for the multiple-uniform-proposals (MUP) WRMC are represented as dashed lines with the same colors (corresponding to same CUDA block size) as the displacement WRMC.

code, thus limiting the length of the random walks. In order to conceive longer random walks, threads from multiple blocks need to be assigned to process the same methane—MFI system, which would effectively replace the one CUDA block per one methane—MFI system mapping that is being currently utilized. In such an implementation, threads from different CUDA blocks would need to be synchronized at each MC step, which can only be achieved through termination of the CUDA kernel given the lack of universal barrier synchronizations in CUDA. Due to the increased latency resulting from relaunching a CUDA kernel at every MC iteration and the diminishing return in statistical accuracy for large thread block sizes, we do not allow for multiple CUDA blocks to process a single system and limit the maximum length of the random walk at 512.

As can be seen from Figure 4, the statistical accuracy in our simulations improves with larger numbers of $n_{threads}$, as expected since more threads contribute to waste recycling. For all values of $n_{threads}$, a small $d_{max}$ (e.g., 0.15 Å) leads to large standard deviation values. While the trial state particle positions near the original position will have reasonably high Boltzmann weights, they are highly correlated with the original position and as such do not add much new information. As $d_{max}$ is increased for all $n_{threads}$, the standard deviations begin to increase again. In this case, while the proposals are more decorrelated from the original position, they are far more likely to have a small Boltzmann weight. As discussed below and in more detail in Appendix , we may capture the scaling of these standard deviations with a simple quantitative model, but regardless of the exact scaling, Figure 4 demonstrates that, in all cases presented here, the standard deviation in WRMC using multiple displacement proposals (MDP) substantially improves over multiple uniform proposals (MUP). The displacement WRMC method becomes equivalent to the uniform WRMC method at $d_{max}$ values equal to dimensions of the simulation box, a distance which is substantially larger than 1.65 Å.

The basic scaling behaviors of the standard deviation as a function of $d_{max}$ and $n_{threads}$ may be quite easily understood from the probability of sampling different distances from the generating point. Three considerations are necessary reflecting the purpose of displacement-based Monte Carlo simulation to gather new *relevant* information by optimizing for sampling near the particle's current position, but not too near that position. (1) Sampling regions too close to the original position of the particle ($r < R_{min}$) does not yield useful additional information, as these positions are within the exclusion zone of the original particle. (2) Sampling regions too far from the original position of the particle ($r > R_{max}$) also does not yield useful information, as these more dispersed locations are less likely to be in the important regions of phase space. (3) Sampling a local region in space with a higher density of points is no longer useful beyond some density $\rho_{cap}$. Disregarding the important effect of $\rho_{cap}$ for the moment, for a fixed number of particles, as the step size increases, initially the number of points between $R_{min}$ and $R_{max}$ increases, leading to better sampling. Then, the number of points begins to decrease again, leading to somewhat poorer sampling of the important nearby region. For a fixed step size, as the number of proposals increases, the number of samples within that window increases in a *nonlinear* fashion, re-emphasizing why the standard deviation decreases as $n_{threads}$ increases. The effect of $\rho_{cap}$ becomes particularly important for large values of $n_{threads}$ and small $d_{max}$, as this combination can lead to a remarkably high number of samples at given radii. Exclusion of point densities exceeding $\rho_{cap}$ is partially responsible for the comparably large standard deviation at small $d_{max}$ even for large $n_{threads}$ as seen in Figure 4.

A random walker model motivated by these very basic considerations of which regions of space are important to sample is developed in Appendix . Hypothetically, this model could allow for the optimal choice of $d_{max}$ and $n_{threads}$ to minimize standard deviation for computation time. However, detailed knowledge of molecular organization in the nonuniform environment is required to yield an accurate model. As such, while we can determine an optimal choice of $d_{max}$ for these systems, in general this is likely not accessible. Therefore, based on the results displayed in Figure 4, we utilize $d_{max}$ = 0.65 Å for the remainder of these studies since that seems reasonably advantageous for all considered $n_{threads}$ and does not rely on optimizing $d_{max}$.

**5.3. Optimal Estimator of Delmas and Jourdain.** We also examine the effect of the optimal estimator on standard deviations. As shown in Figure 5, for both 32 threads and 128 threads, the linear combination of the traditional MC and WRMC results via the factor $b^*$ yields a reduction in the standard deviation of the energy. The estimation of $b^*$ itself does come at a small computational cost; however, the benefits in standard deviation decrease outweigh this computational cost. The optimal estimator appears to have a more beneficial effect for simulations with greater numbers of threads. This likely is a consequence of the improved estimation of $b^*$ itself as more terms are included in the waste-recycling average. Since the use of the optimal estimator provides enhanced accuracy of prediction with minimal computational cost, all further results for multiproposal WRMC shall employ the optimal estimator. The values of $b^*$ are shown in Table 1.

**5.4. Optimization of Thread Block Configuration.** We now begin to optimize our CUDA configuration. In the previous

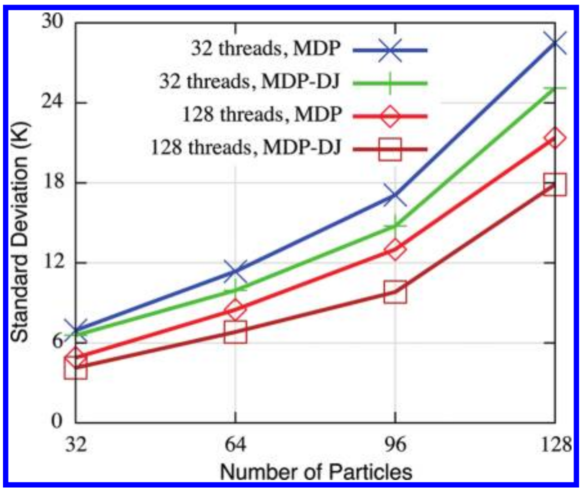


Figure 5. Standard deviation of the average energy as a function of the number of particles for $n_{\text{threads}} = 32$ and 128 for the multiple-displacement-proposals algorithm with (MDP-DJ) and without (MDP) the optimal estimator for 1 million accumulation steps.

**Table 1. Values of $b^*$ for MDP-DJ, Which Are Found Taking the Averages of $\langle E \rangle$, $\langle E^2 \rangle$, and $\langle (E_{m+1} - E_m)^2 \rangle$ from the 14 CUDA Blocks, Which Processes Independent MFI Frameworks**

| | number of particles ($N_{\text{tot}}$) | | | |
|---|---|---|---|---|
| $n_{\text{threads}}$ | 32 | 64 | 96 | 128 |
| 32 | 66.74 | 136.20 | 224.92 | 400.40 |
| 128 | 43.14 | 89.93 | 162.07 | 321.72 |

displacement WRMC simulation results, the total number of CUDA blocks is fixed at 14 with one streaming multiprocessor executing a single CUDA block. This mapping prevents the GPU hardware from scheduling warps from different thread blocks and leads to hardware underutilization, which is especially damaging for small $n_{\text{threads}}$ at low occupancy. In order to remedy this situation and to determine the optimal block size/number, we increase the number of CUDA blocks and run simulations with different combinations of CUDA block size (denoted as $n_{\text{threads}}$) and the number of CUDA blocks per multiprocessor (denoted as $n_{\text{blocks}}$). The single iteration wall time numbers for CUDA thread configurations ($n_{\text{threads}}$, $n_{\text{blocks}}$) = (32, 16), (64, 8), (128, 4), (256, 2), and (512, 1) are plotted in Figure 6 for $N_{\text{tot}}$ = 32, 64, 96, and 128 methane molecules. In all of the simulations, $n_{\text{blocks}} \times n_{\text{threads}} = 512$, and the total number of CUDA threads is fixed at 512 × 14 = 7168. The total number of independent methane−MFI system is $14n_{\text{blocks}}$.

From Figure 6, it can be seen that the single iteration time decreases from the configuration (32, 16) to (64, 8) for all $N_{\text{tot}}$. Because the maximum resident number of thread blocks per multiprocessor is limited to eight in Fermi Tesla C2050 cards, only eight out of the 16 $n_{\text{blocks}}$ can concurrently occupy a streaming multiprocessor in (32, 16), resulting in an occupancy number of only 256/1536 = 0.166 as opposed to 512/1536 = 0.333 for other configurations, which all have eight or less $n_{\text{blocks}}$. The GPU hardware can be more fully utilized in configuration settings with a higher occupancy number, and subsequently we observe the wall time reduction from (32, 16) to (64, 8).

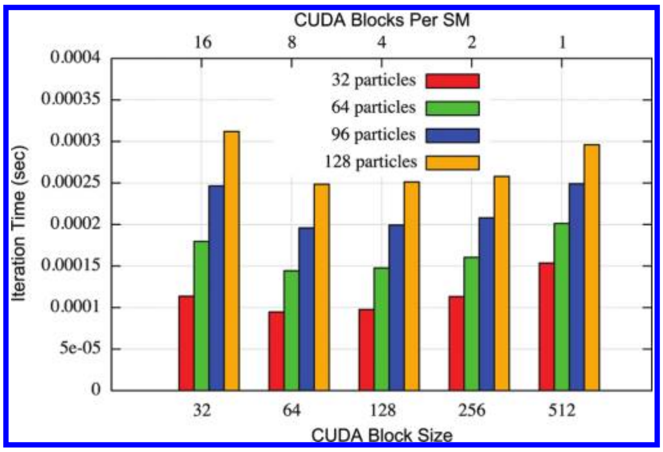


Figure 6. Iteration time as a function of CUDA block size and number of blocks per streaming multiprocessor for 32, 64, 96, and 128 particles (from left to right, respectively) using the MDP-DJ algorithm. The total number of CUDA blocks is $14n_{\text{blocks}}$.

For larger $n_{\text{threads}}$, the wall time increases from (64, 8) as the proportional time spent in the kernel that generates the length $n_{\text{threads}}$ random walks becomes larger. Because threads having different $n_{\text{blocks}}$ generate the random walks in parallel in our implementation, only the value of $n_{\text{threads}}$ largely determines the kernel wall time. Overall, these two factors work together to make the configuration ($n_{\text{threads}}$, $n_{\text{blocks}}$) = (64, 8) possess the shortest wall time for all $N_{\text{tot}}$.

It is important to keep in mind that we cannot easily determine the optimal block/thread size by comparing just the iteration wall times since each configuration possesses different values of $n_{\text{threads}}$ and provides varying statistical accuracy. In order to meaningfully evaluate the best ($n_{\text{threads}}$, $n_{\text{blocks}}$) configurations, we follow the same strategy used to derive results in Figure 3b and set the accumulation steps for (32, 16) to be a baseline number of 1 million and adjust the number of steps in other configurations to equate the total wall time spent in the GPU for all ($n_{\text{threads}}$, $n_{\text{blocks}}$). Again, 40 independent displacement WRMC simulation runs that include the optimal estimator are conducted, and the average energy and standard deviation value are tabulated in Table 2 (the values for $n_{\text{blocks}}$ are the same ones used in Figure 6 and omitted in the labels). For comparison, we include simulation results from parallel Lennard-Jones and embarrassingly parallel MC methods with equal wall time for all $N_{\text{tot}}$, and all of the results are plotted in Figure 7 to illustrate the behavior. In the parallel Lennard-Jones method, the (32, 16) configuration was utilized for all $N_{\text{tot}}$, as this results in minimum single iteration wall time. In the embarrassingly parallel MC method, we used the (64, 8) configuration, which also provided the minimum single iteration wall time. For all $N_{\text{tot}}$, the lowest standard deviation values among the displacement WRMC methods are observed for the (32, 16) and the (64, 8) configurations, as the performance cost of reducing the number of independent ensembles outweighs the benefit of increasing the number of WRMC proposal trial states for $n_{\text{threads}} > 64$. Overall, similar to results found from the nonoptimized configurations in Figure 3b, the overall most accurate results were found from the embarrassingly parallel MC algorithm.

**5.5. Comparison of GPU to CPU Timing.** We also compare the efficiency of the GPU methods to that of a single core CPU monoproposal MCMC method. The algorithm for CPU MCMC

Table 2. Average Energy and Standard Deviation Values for 32, 64, 96, and 128 Particles for Different $n_{\mathrm{threads}}$ in Units of Kelvin for the MDP-DJ Algorithm

| | number of particles ($N_{\mathrm{tot}}$) | | | |
|---|---|---|---|---|
| $n_{\mathrm{threads}}$ | 32 | 64 | 96 | 128 |
| 32 | −68354.59 ± 1.39 | −138027.62 ± 2.97 | −209278.78 ± 3.24 | −281343.57 ± 7.63 |
| 64 | −68353.49 ± 1.27 | −138027.38 ± 2.56 | −209278.40 ± 3.21 | −281340.17 ± 7.58 |
| 128 | −68354.19 ± 1.67 | −138027.66 ± 2.27 | −209280.23 ± 3.73 | −281340.30 ± 9.55 |
| 256 | −68355.18 ± 1.97 | −138028.18 ± 3.50 | −209278.08 ± 5.77 | −281343.87 ± 10.42 |
| 512 | −68352.43 ± 3.32 | −138028.45 ± 4.84 | −209277.23 ± 7.04 | −281344.35 ± 13.05 |

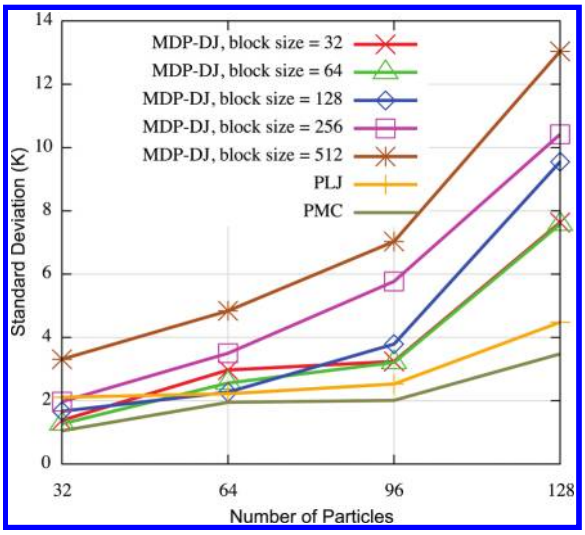


Figure 7. Standard deviation in energy for the multiple-displacement-proposals (MDP-DJ) WRMC results from Table 2 as well as parallel Lennard-Jones (PLJ) and embarrassingly parallel simulations (PMC). The block sizes of 32 and 64 (i.e., $n_{\mathrm{threads}} = 32$ and 64) provide the lowest standard deviation for the MDP-DJ algorithm.

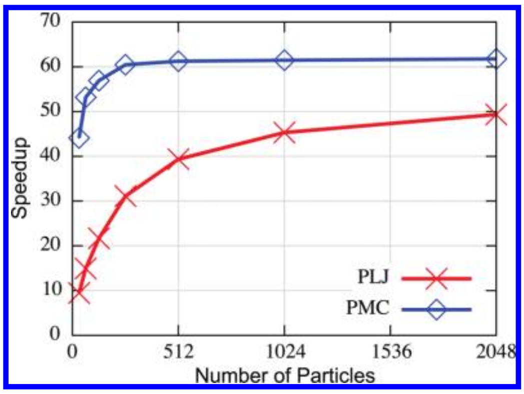


Figure 8. Speedup as a function of the number of particles for system sizes from 32 to 2048 methane molecules for double-precision (DP) calculations using the Fermi Tesla C2050 card. The speedup is defined as a ratio between single iteration CPU and a single iteration GPU wall time. The maximum speedup of 61.75 is observed for 2048 methane molecules.

is similar to the one used in the parallel Lennard-Jones method except for the serial processing of the pair-potential calculation. The CPU MCMC source code is compiled using an Intel 11.1 compiler, which provides faster CPU single iteration time compared to gcc 4.4.2 for small system sizes. We define speedup as the ratio between the CPU and the GPU single iteration times and plot the results in Figure 8. It is difficult to compare the single iteration wall times with the WRMC algorithm and arrive at any meaningful conclusion since the two methods collect samples with uneven importance weights. Accordingly, we choose to omit this comparison and concentrate on GPU results from the parallel Lennard-Jones and the embarrassingly parallel methods. Unlike previous simulations, we simulate relatively large system sizes that provide unphysical high loading situations to better assess the speedup behaviors. Accordingly, the energy results are erroneous for large system sizes, which is acceptable in this context since only the numerical speedup results are meaningful and of interest here. As can be seen from Figure 8, the embarrassingly parallel MC method has better performance over the parallel Lennard-Jones method for all system sizes with a maximum speedup value of 61.75 at 2048 methane molecules. However, for even larger system size, we expect the two methods to provide similar values of speedup as the overhead from routines involved in tasks other than pair-potential calculation become negligible for the parallel Lennard-Jones method, which is the cause of slower speedup in small systems. For particle loadings of interest in the methane−MFI system, at 32, 64, and 128 particles, there are respectively 9.47 (44.11), 14.93 (53.18), and 21.68 (56.90) speedups in the parallel Lennard-Jones (embarrassingly parallel) method over the CPU results.

**5.6. Application to Free Energy Calculation.** Finally, we consider a simulation scenario where waste recycling was shown to be useful in the literature—the calculation of the free-energy profile along a reaction coordinate. The forms of $P(z)$ and $\beta F(z)$ shown in Figure 9 nicely illustrate the simple free-energy barrier in the windows between two cages, with mild corrugation within the cages due to the packing of those methane molecules occupying the cages.

In the calculation of these histograms $P(z)$, the previous efforts in this paper toward optimization on the GPU are portable, so we determine $P(z)$ simply using the GPU-optimized mappings. *A priori*, we expect this to be a scenario where waste recycling Monte Carlo could lead to faster convergence in the histogram because the higher free-energy states within the window between cages will be sampled by the Markov chain of states much less frequently, yet some subset of the multiple proposals not accepted likely probes this region of space. And indeed, our expectations are bourne out. For simulations of 32 methane molecules in the zeolite LTA for 10 million MC cycles, $P(z)$ is calculated via simple binning. In this instance, both parallel Lennard-Jones (PLJ, not shown) and waste-recycling with multiple displacement proposals (MDP, Figure 9) give quite similar forms for $P(z)$.

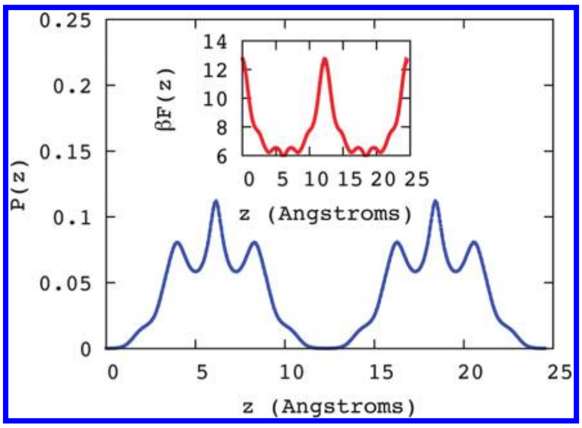


**Figure 9.** Plots for $P(z)$ vs $z$ (blue) and $\beta F(z)$ vs $z$ (red, inset) for LTA with 32 methane molecules (10 million Monte Carlo cycles using the waste recycling with multiple displacement proposals (MDP) with $n_{\text{threads}} = 32$ and $n_{\text{blocks}} = 8$). $P(z)$ indicates the histogram of probabilities, while $\beta F(z)$ represents the free-energy (unitless) along the $z$ direction of the LTA (unit cell length =24.555 Å). The free-energy barrier is graphically represented by the bump near $z = 0.5 \times 24.555$ Å in the inset curve.

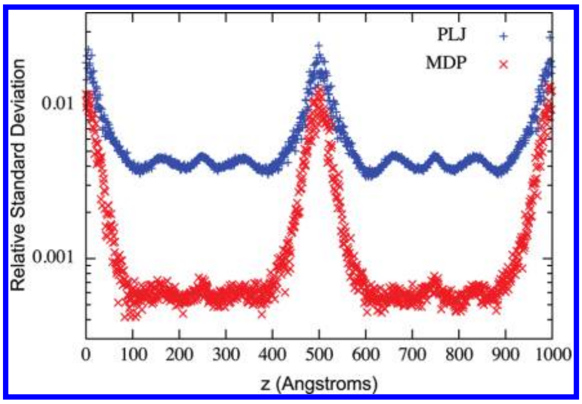


**Figure 10.** Relative standard deviation as a function of position across the LTA zeolite for PLJ and MDP simulations with equal wall times.

To assess timing gains, we run simulations using both techniques such that the total wall time is fixed (10 million MDP cycles and 51.2 million PLJ cycles). For the MDP simulations, we exclude the optimal estimator of Delmas and Jourdain as the system size (e.g., 32 methane molecules) is sufficiently small that the overhead required in obtaining relevant quantities would most likely not decrease the standard deviation taking into account wall time. Moreover, the main point here is to emphasize the performance difference between the waste recycling method and other more conventional parallel methods. We consider standard deviations across a set of 20 MC simulations for each parallelization strategy. In order to better quantify any computational gains, we consider a single point in these normalized probability distributions, $P(z^*)$. We choose $z^*$ to be the least probable location, with $z^*$ thereby corresponding to the free energy maximum, and $P(z^*)$ is a quantity important in determining diffusion rates through zeolites.[17] For the system of 32 methane molecules in LTA, parallel LJ yields $2.41 \times 10^{-4} \pm 4.04 \times 10^{-6}$, while waste recycling Monte Carlo with multiple displacement proposal yields $2.44 \times 10^{-4} \pm 1.13 \times 10^{-6}$. In Figure 10, we show the relative standard deviation (i.e., standard deviation divided by the mean value) at all of the bins across a single dimension of the LTA zeolite for the same sets of simulations. As can be seen, the MDP outperforms PLJ across the entire set of bins. Thus, as hypothesized, waste recycling Monte Carlo on a GPU is computationally advantageous, oversimply accelerating the Lennard-Jones potential calculation when sampling *rare* events.

## 6. CONCLUSIONS

Waste recycling Monte Carlo has previously been proposed as a way to leverage the "unused" rejected trial states in Monte Carlo simulations. Two examples of scenarios where waste recycling has proven particularly useful are (1) parallel tempering where there exists a large amount of fallow data available for immediate harvesting and (2) calculation of potentials of mean force which rely on histogram collection.[8,17,18]

Here, we examine in detail the application of WRMC to the calculation of the simple average of energy $E$ in canonical Monte Carlo simulations, where waste recycling is not particularly beneficial for single trial moves. Such an approach has allowed us to optimize the GPU implementation of several distinct parallelization strategies. The strategies are (1) embarrassingly parallel Monte Carlo (PMC), (2) parallel Lennard-Jones calculation (PLJ), (3) waste recycling based on multiple uniform proposals (MUP), and (4) waste recycling based on multiple displacement proposals (MDP). Our figure of merit in these studies of $\langle E \rangle$ has been the uncertainty in this average for each technique, with a fixed computational wall time.

Our analysis of various GPU-implementation optimizations for calculating the average energy has found that the use of parallel Monte Carlo simulations is the most computationally efficient approach. Such a conclusion is not surprising because the sampling of $n_{\text{threads}}$-fold more independent simulation systems than for PLJ or WRMC leads to a greater reduction in variance. However, simply by virtue of limited memory resources, such an embarrassingly parallel approach to MC simulation will not always be feasible.

Allocating one simulation to an entire thread block eases those memory constraints while leading to fewer total simulations. We examined two possible approaches for using the computational resources within a thread block—parallelization of the Lennard-Jones pair potential calculation and the evaluation and waste recycling of multiple Monte Carlo proposals. Among these techniques, PLJ is the most efficient computational approach for evaluating the average energy, allowing for the fastest propagation through phase space based on CPU wall time. In essence, this again is simply because efficient propagation through phase space via the PLJ algorithm dominates for variance reduction in the average energy.

The effectiveness of the multiple-uniform-proposals WRMC algorithm decreases with greater density of particles as the trial states generated from the uniformly random distribution provide high energy states that make negligible contributions to the statistics. Generating trial states based on displacements, as done in the multiple-displacement-proposals WRMC algorithm, sidesteps this problem. With careful generation of displacement trial states based on a random walk of displacements, very little modification of the remainder of the WRMC algorithm is required.

When calculating simple MC averages such as $\langle E \rangle$, parallel Monte Carlo simulations with one simulation per thread (PMC) is the algorithm of choice, followed by parallelization of the Lennard-Jones calculations (PLJ) within a thread block. Neither WRMC algorithm is as computationally efficient. However, as noted earlier, in a variety of scenarios related to determination of histograms and sampling of rare events, WRMC can be advantageous on the CPU even when it is not for other quantities.

In particular, the free energy profile of particles in the caged zeolite structures, crucial for characterizing diffusive behavior, is straightforwardly related to the density profile of methane molecules in the zeolite LTA. Therefore, we also compare the optimized version of each of these algorithms for determination of the free-energy profile between adsorption cages within the zeolite LTA, in order to assess if WRMC on the GPU is then worthwhile. Indeed, WRMC with multiple displacement proposals (MDP) is more computationally efficient than either PMC or PLJ in conducting free energy calculations.

Beyond assessing various Monte Carlo algorithms for framework-adsorbate systems, we have also examined the utility of the optimal estimator of Delmas and Jourdain[15] in a multiproposal scenario for the first time. We find that this optimal estimator is advantageous for energy calculations with relatively little computational overhead and provides meaningful improvement in the estimation of properties.

While our studies indicate that often multiproposal waste-recycling algorithms are not computationally the ideal path for employing GPU resources in MC simulations, waste recycling can be beneficial over other parallelization strategies for sampling rare events. Our findings suggest that the relative merits of WRMC in a parallel GPU computing environment are comparable to those in a single CPU computing environment, with waste-recycling Monte Carlo advantageous for sampling rare events but less useful for straightforward MC averages. However, the careful implementation of a multiproposal framework as developed in this paper is necessary to even employ waste recycling in the GPU environment. Furthermore, we are currently expanding the work on the other GPU implementations (PMC and PLJ) in order to substantially accelerate calculation of adsorption isotherms in zeolites, as this is necessary to computationally screen the millions of hypothetical zeolite structures.[23]

## APPENDIX A. COMPILER AND PROCESSOR DESCRIPTION

The NERSC cluster used, Dirac, is a testbed GPU cluster that contains 44 Fermi Tesla C2050 GPUs, which come equipped with 448 CUDA cores, 14 streaming multiprocessors (SMs), 3 GB of DRAM, and ECC memory. The card delivers peak single-precision (double-precision) performance of 1.03 TFlops (515 GFlops) and 144 GB/s of peak memory bandwidth. The PCI Express 2.0 with 16 lanes is used to transfer data back and forth from the CPU to the GPU memory. The CPU node within Dirac consists of two Intel 5530 2.4 GHz, quad core Nehalems with an 8 MB cache, 5.86 GT/s QPI, and 24 GB DDR3-1066 Reg ECC memory. The CUDA compiler driver NVCC along with gcc 4.4.2 with -O3 optimization flag is used for all of the GPU simulations, whereas for CPU simulations, the Intel C++ Compiler 11.1 is used. CUDA Toolkit 3.2 along with CUDA C runtime is used, and the CURAND library is utilized to generate pseudorandom numbers based off of the XORWOW algorithm.[19] The random numbers are generated directly inside the device kernel using the device API, thereby bypassing the need to transfer the numbers from the CPU to the GPU. Random generator state initialization (curand_init()) and random number generation (curand()) are divided into two separate kernels in order to maximize performance. All of the results reported in this work use double-precision (64-bit) floating point numbers.

## APPENDIX B. INCORRECT OR INEFFICIENT MULTIPLE DISPLACEMENT PROPOSALS WITH WASTE-RECYCLING

In section 4.4, we described our chosen MDP algorithm with an emphasis placed on the generation of trial positions such that we may use the simple symmetric multiproposal Barker-like acceptance ratio. Here, we describe alternative routes that proved either incorrect or inefficient.

**Incorrect MDP Algorithms.** A naive, and incorrect, modification of the multiproposal waste recycling algorithm in order to generate multiple displacement proposals would involve generating the proposed particle positions all as single uniform displacements from the original position of particle $k$. Such an approach yields incorrect results. This stems from the fact that not all points within the set $P = \{o,\{n\}\}$ are accessible to each other via this trial-state generation algorithm. Instead, generating $\{n\}$ based on drawing displacements from a random Gaussian distribution removes this formal accessibility barrier since there is always some finite possibility of any point in $A$ generating the remaining points. However, this set $P$ still has inherent bias towards the state $o$ as the generating point. Delmas and Jourdain[15] define a multiproposal acceptance ratio accounting for the *a priori* probabilities of trial generation based on the associated Gaussian probability density that corrects this state generation bias. However, for the range of numbers of proposals explored in this paper, calculated properties still exhibited a dependence on the number of proposals.

**Inefficient MDP with Configurational Bias.** The crux of our employed multiproposal algorithms lies in constructing a set of proposals that are equally likely to have proposed that set, and this results in a simply symmetric Boltzmann acceptance ratio. However, a more traditional approach to multiproposal Monte Carlo would be the implementation of configurational bias Monte Carlo (CBMC).[7] A standard CBMC algorithm for multiple proposals might propose a set of trial states $\{n\}$ based on displacements from the old state $o$. A proposed state is then chosen solely from the set of trial states $\{n\}$ based on their relative Boltzmann weights. Once a proposed state is chosen, it is either accepted or rejected on the basis of both the sum of the Boltzmann weights from the set of trial states $\{n\}$ and a sum of Boltzmann weights due to a hypothetical set of trial states $\{o'\}$ which are generated by displacements from the proposed state. This generation of forward and backward sets of trial states is required in order to fulfill the condition of superdetailed balance.[7]

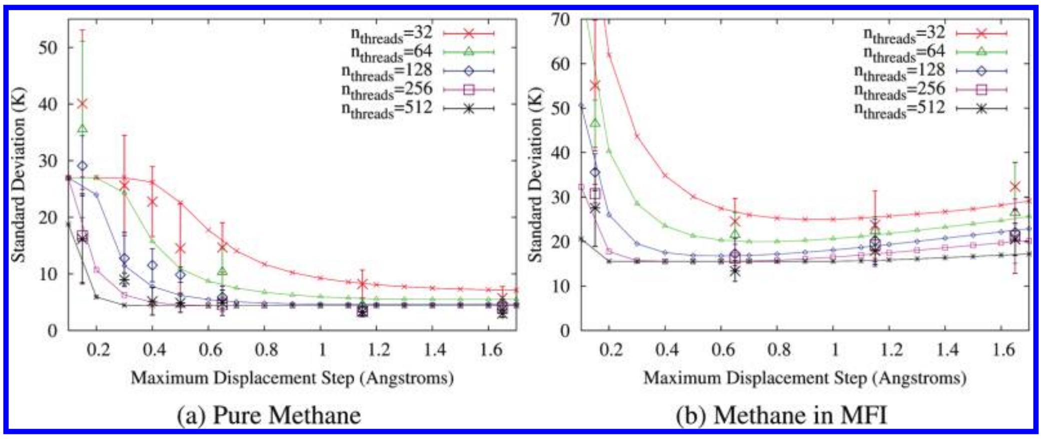
Simulating with multiproposal configurational bias yields correct results; however, optimal implementation of the waste recycling approach would require the accumulation of the following summation:

$$A = \sum_{i \in \{n\}} \frac{w_i}{W}\{A_i \operatorname{acc}(o \rightarrow i) + A_o[1 - \operatorname{acc}(o \rightarrow i)]\} \tag{9}$$

where $w_i$ is defined again as $\exp(-\beta E_i)$ and $W$ is defined as

$$W = \sum_{i \in \{n\}} w_i \tag{10}$$

with the old state *excluded* from the summation. Aside from this exclusion, eq 10 is similar to weightings employed thus far for waste recycling.



(a) Pure Methane

(b) Methane in MFI

**Figure 11.** Random walker model of standard deviation in calculated energies for both 128 methane molecules as a uniform fluid (a) and 128 methane molecules in the zeolite MFI (b). Each is modeled via the proportionality in eq 12 with a single proportionality constant $A$ for all $n_{\text{threads}}$ and $d_{\max}$. For the uniform methane fluid, $A = 27.0$, $\rho_{\text{cap}}\sigma^3 = 1$, $R_{\min} = 3.8$ Å, and $R_{\max} = 8.0$ Å. For methane adsorbed in MFI, $A = 110.0$, $\rho_{\text{cap}}\sigma^3 = 10$, $R_{\min} = 1.0$ Å, and $R_{\max} = 4.0$ Å. Choice of these parameters is discussed in the text. For each plot, the data from GPU calculations are displayed using the larger symbols with error bars, and the results from the simple random walker model are presented with the smaller symbols and lines to guide the eye.

The inefficiency in implementing waste recycling for configurational bias lies in the definition of acceptance probabilities. In order to determine an acceptance ratio for a specific trial state $i$, we must generate a set of trial old states $\{o'|i\}$ from the state $i$ in order to obey superdetailed balance. Given these *unique* sets of old trial states, the acceptance probability for any state $i$ is defined as

$$\text{acc}(o \rightarrow i) = \min\left[1, \frac{W}{W(\{o'|i\})}\right] \tag{11}$$

Since a different set $\{o'|i\}$ must be generated for each proposed trial state $i$, a total of $N_{\text{prop}}(N_{\text{prop}} - 1)$ new particle positions and new energies must be evaluated in order to fully leverage waste recycling coupled to multiproposal MC, instead of $N_{\text{prop}}$ energies, as is the case for our MDP algorithm. A waste recycling expression similar to the above CBMC algorithm has been successfully and efficiently employed by Athénes and Calvo[18] in the context of replica exchange. However, in replica exchange simulations, the associated acceptance ratios require no new calculations of energies and simply require the rescaling of previously determined energies by new factors $\beta$.

Our proposed approach for multiproposal waste recycling involving the construction of random walks, as stated in subsection 4.3, yields correct averaged results which are invariant to the number of proposals. Furthermore, it allows inclusion of all calculated energies into the waste-recycling expression for accumulating averages. In the implementation based on constructing a random walk chain, no calculated energies lie fallow. As an added benefit, since this approach is constructed via Barker-like acceptance ratios rather than Metropolis-like acceptance ratios, we may employ the optimal estimator discussed in section 4.5.

## APPENDIX C. SIMPLE MODEL FOR MDP PARAMETERS

The variance in the average energy for the MDP waste-recycling algorithm has a rather unusual functional form, as shown in Figure 4. With the considerations outlined in section 5.2, we may quantitatively model the variation in standard deviation as a function of $d_{\max}$ and $n_{\text{threads}}$. We display our model results in Figure 11 for both the standard deviations for methane in MFI and for pure methane. For each value of $d_{\max}$ and $n_{\text{threads}}$, we determine the number densities associated with finding a position on the random walk a certain distance $r$ from the origin. These densities are calculated on the basis of a total of $10^6$ distinct random walks constructed numerically following the algorithm employed in this paper for the WRMC simulations and accounting for periodicity as the paths are constructed. Each number density $N(r;d_{\max},n_{\text{threads}})$ is initially calculated such that $\int_0^\infty N(r;d_{\max},n_{\text{threads}})\,\mathrm{d}r = n_{\text{threads}} - 1$ and is subsequently capped at each $r$ to be at maximum $\rho_{\text{cap}}4\pi r^2$. This $N(r;d_{\max},n_{\text{threads}},\rho_{\text{cap}})$ thereby encompasses effects due to the spherical geometry at each $r$ as well as the limits on the effectiveness of higher sampling number density. The standard deviation on a calculation is hypothesized to scale inversely with the square root of the number of meaningfully sampled points. We express this as

$$\text{std.dev.} \propto \frac{1}{\sqrt{1 + \int_{R_{\min}}^{R_{\max}} N(r;\, d_{\max}, n_{\text{threads}}, \rho_{\text{cap}})\,\mathrm{d}r}} \tag{12}$$

where the original position is always included as a meaningful point and all other sampled points are solely included if they are between $R_{\min}$ and $R_{\max}$ and have not exceeded the local density $\rho_{\text{cap}}$.

As shown, in Figure 11, this basic model captures semiquantitatively the features of the standard deviation of energy as a function of $d_{\max}$ and of $n_{\text{threads}}$ for both a uniform Lennard-Jones fluid of 128 methane molecules with the MFI framework removed (corresponding to $\rho\sigma^3 = 0.156$) and for 128 methane molecules adsorbed in eight unit cells of the zeolite MFI. For methane as a uniform LJ fluid, $\rho_{\text{cap}}$ is set by the relationship $\rho_{\text{cap}}\sigma^3 = 1$, and $R_{\min}$ and $R_{\max}$ are set to 3.8 Å and 8.0 Å, approximately $\sigma$ and $2\sigma$, perfectly reasonable parameters for a moderately dense gas. For methane adsorbed in the zeolite MFI, we used substantially different parameters based on knowledge of the organization of methane in MFI. For this degree of adsorption, the methane sites are separated by approximately 5 Å, and within the channels, they organize in a single file. As such, we set $R_{\min}$ to be 1.0 Å in order to account for meaningful sampling within the radius of the MFI channel. And we choose $R_{\max}$ to be 4.0 Å since distances greater than this yield configurations overlapping with their nearest neighbors. Given that the highly packed and nonuniform nature of this system introduces greater corrugations, we choose $\rho_{cap}$ to be 10 times higher. The agreement between our simple random walker model and the GPU calculated standard deviations is quite favorable for both the uniform and nonuniform systems, as displayed in Figure 11.

## AUTHOR INFORMATION

### Corresponding Author

*E-mail: jihankim@lbl.gov; Berend-Smit@berkeley.edu.

## ACKNOWLEDGMENT

We thank Joseph Swisher and Mahmoud Forrest Abouelnasr for useful discussion and guidance in simulating zeolites. This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under contract no. DE-AC02-05CH11231. B.S. was supported as part of the Center for Gas Separations Relevant to Clean Energy Technologies, an Energy Frontier Research Center funded by the U.S. Department of Energy, Office of Science, Office of Basic Energy Sciences under Award Number DE-SC0001015. J.K. was supported by the Director, Office of Science, Advanced Scientific Computing Research, of the U.S. Department of Energy under contract no. DE-AC02-05CH11231. J.M.R. acknowledges the support of the Chemical Sciences, Geosciences and Biosciences Division, Office of Basic Energy Sciences, Office of Science, U.S. Department of Energy, FWP number SISGRKN.

## REFERENCES

(1) Frenkel, D. *Proc. Natl. Acad. Sci. U.S.A.* **2004**, *101*, 17571–17575.

(2) NVIDIA CUDA Programming Guide 3.0. http://developer.download.nvidia.com/compute/cuda/3_0/toolkit/docs/NVIDIA_CUDA_ProgrammingGuide.pdf (accessed August 29, 2011).

(3) (a) Anderson, J.; Lorenz, C.; Travesset, A. *J. Comput. Phys.* **2008**, *227*, 5342–5359. (b) Stone, J.; Phillips, J.; Freddolino, P.; Hardy, D.; Trabuco, L.; Schulten, K. *J. Comput. Chem.* **2007**, *28*, 2618–2640.

(4) Preis, T.; Virnau, P.; Paul, W.; Schneider, J. *J. Comput. Phys.* **2009**, *228*, 4468–4477.

(5) Li, H.; Petzold, L. *Int. J. High Perform. Comput. Appl.* **2010**, *24*, 107–116.

(6) Anderson, A.; Goddard, W.; Schroder, P. *Comput. Phys. Commun.* **2007**, *177*, 298–306.

(7) Frenkel, D.; Smit, B. *Understanding Molecular Simulation: From Algorithms to Applications*, 2nd ed.; Academic Press: San Diego, CA, 2002.

(8) Athènes, M.; Marinica, M.-C. *J. Comput. Phys.* **2010**, *229*, 7129–7146.

(9) Smit, B.; Maesen, T. L. M. *Chem. Rev.* **2008**, *108*, 4125–4184.

(10) Smit, B. *J. Phys. Chem.* **1995**, *99*, 5597–5603.

(11) Dubbeldam, D.; Calero, S.; Vlugt, T.; Krishna, R.; Maesen, T.; Beerdsen, E.; Smit, B. *Phys. Rev. Lett.* **2004**, *93*, 088302.

(12) Dubbeldam, D.; Calero, S.; Maesen, T.; Smit, B. *Phys. Rev. Lett.* **2003**, *90*, 245901.

(13) Dubbeldam, D.; Beerdse, E.; Vlugt, T.; Smit, B. *J. Chem. Phys.* **2005**, *122*, 224712.

(14) Dubbeldam, D.; Calero, S.; Vlugt, T.; Krishna, R.; Maesen, T.; Smit, B. *J. Phys. Chem. B* **2004**, *108*, 12301–12313.

(15) Delmas, J.-F.; Jourdain, B. *J. Appl. Probab.* **2009**, *46*, 938–959.

(16) Beerdsen, E.; Dubbeldam, D.; Smit, B. *J. Phys. Chem. B* **2006**, *110*, 22754–22772.

(17) Coluzza, I.; Frenkel, D. *ChemPhysChem* **2005**, *6*, 1779–1783.

(18) Athènes, M.; Calvo, F. *ChemPhysChem* **2008**, *9*, 2332–2339.

(19) CUDA CURAND Library. http://developer.download.nvidia.com/compute/cuda/3_2/toolkit/docs/CURAND_Library.pdf (accessed August 29, 2011).

(20) Metropolis, N.; Rosenbluth, A.; Rosenbluth, M.; Teller, A.; Teller, E. *J. Chem. Phys.* **1953**, *21*, 1087–1092.

(21) NVIDIA CUDA C SDK. http://developer.nvidia.com/cuda-toolkit-sdk (accessed August 29, 2011).

(22) Adjanor, G.; Athènes, M.; Rodgers, J. M. *J. Chem. Phys.* **2011**, *135*, 044127.

(23) Earl, D.; Deem, M. *Ind. Eng. Chem.* **2006**, *54*, 5449–5454.