Computer Physics Communications 183 (2012) 2054–2062

Contents lists available at SciVerse ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

Massively parallel chemical potential calculation on graphics processing units

Kevin B. Daly, Jay B. Benziger, Pablo G. Debenedetti, Athanassios Z. Panagiotopoulos *

Department of Chemical and Biological Engineering, Princeton University, Princeton, NJ 08544, United States

ARTICLE INFO

Article history:
Received 6 March 2012
Received in revised form
1 May 2012
Accepted 3 May 2012
Available online 11 May 2012

Keywords:
Monte Carlo methods
Phase equilibria
Graphics processing units
Free energy

ABSTRACT

One- and two-stage free energy methods are common approaches for calculating the chemical potential from a molecular dynamics or Monte Carlo molecular simulation trajectory. Although these methods require significant amounts of CPU time spent on post-simulation analysis, this analysis step is well-suited for parallel execution. In this work, we implement this analysis step on graphics processing units (GPUs), an architecture that is optimized for massively parallel computation. A key issue in porting these free energy methods to GPUs is the trade-off between software efficiency and sampling efficiency. In particular, fixed performance costs in the software favor a higher number of insertion moves per configuration. However, higher numbers of moves lead to lower sampling efficiency. We explore this issue in detail, and find that for a dense, strongly interacting system of small molecules like liquid water, the optimal number of insertions per configuration can be as high as 10^5 for a two-stage approach like Bennett's method. We also find that our GPU implementation accelerates chemical potential calculations by as much as 60-fold when compared to an efficient, widely available CPU code running on a single CPU core.

1. Introduction

Calculating differences in free energy between two states of a system is an important task for statistical physics. These systems can be as diverse as proteins, ligands, hygroscopic membranes, and coexisting vapor and liquid phases [1–3]. Free energy calculations typically begin by sampling configurations in a convenient ensemble using either molecular dynamics or Monte Carlo simulations. These configurations may come from the initial state, the final state, or one of many intermediate states, depending on the method of calculating free energy. Some methods also require extensive post-simulation analysis of these configurations, in particular methods involving many trial insertions and deletions. The CPU time required by this analysis may exceed that of generating the configurations, depending on the system and the method. Therefore, in order to minimize the total CPU time, the method should be chosen carefully.

In the test-particle insertion (TPI) method [4], much of the computational burden is shifted to post-simulation analysis. The calculated quantity is the excess chemical potential, that is to say the free energy difference between systems compared with N and $N+1$ particles, and the corresponding quantity in an ideal gas (non-interacting) state, holding other independent thermodynamic variables fixed. The method consists of performing random, independent test insertions in configurations of the N-particle initial state and computing the resulting energy changes. The statistical mechanical expression linking appropriate averages of the energy change upon attempted insertion and the chemical potential is given in Section 2. Multiple insertions may be performed per configuration, resulting in a trade-off between the CPU time needed to generate the configurations and that required for the post-simulation analysis. Even after minimizing the total CPU time, this method can still be impractical for dense, strongly interacting systems like liquid water, as we demonstrate in Section 4.

A two-stage technique, like Bennett's method [5], can greatly improve sampling efficiency for these difficult systems. Like the TPI method, Bennett's method involves randomly inserting test molecules in the N-particle system. In addition, it performs test deletions from the $N+1$-particle system. It then optimally combines the energy changes from these two types of moves in order to minimize the variance of the free energy estimate. The relevant statistical mechanical equations are given in Section 2. A crucial assumption of this method is that test insertions and deletions are uncorrelated. Therefore, care must be taken when performing many test moves in the same configuration. So far, much of the published work on Bennett's method involves a single insertion and deletion per configuration [6–8], shifting the computational burden to generating configurations.

A method that relies even more heavily on generating configurations is thermodynamic integration. This approach involves simulations at many intermediate states, but does not require any post-simulation test moves. Instead, it estimates the free energy

* Corresponding author.
E-mail address: azp@princeton.edu (A.Z. Panagiotopoulos).

0010-4655/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.cpc.2012.05.006

based on the ensemble averages of the potential energy for all of the state points. Like Bennett's method, it has been successfully applied to dense, strongly-interacting systems [2]. Note that both Bennett's and the OS method are still limited to systems of small molecules, in which there is appreciable overlap between the distributions of insertion and deletion energies. For larger molecules, a method involving more than two stages may be necessary [9].

A key advantage of shifting the computational burden to the post-simulation analysis is that this step is usually straightforward to implement on parallel computer architectures. In contrast, parallelizing Monte Carlo is much more difficult. While molecular dynamics algorithms have been efficiently parallelized, diminishing returns in performance are seen when executed over large numbers of CPU cores [10]. The post-simulation analysis can be parallelized in multiple ways for the TPI and Bennett's method. For example, one may divide configurations from a single simulation among different CPUs. Alternatively, one may divide test insertions/deletions for a given configuration among multiple CPUs.

This second parallel implementation of the post-simulation analysis would seem to be the least attractive, given that the other method is simpler to implement. However, it is an appropriate implementation for graphical processing units (GPUs), a parallel architecture with a rapidly growing number of applications in scientific computing [11]. GPUs consist of hundreds of cores executing floating point operations in parallel, leading to impressive performance gains over a single CPU for certain types of particle simulations [10]. However, GPUs have $O(10\ \text{MB})$ of global memory available per core, much less than the $O(1\ \text{GB})$ of RAM available per core in a typical CPU cluster. Therefore, GPUs may not be capable of simultaneously storing data for many configurations. This memory limitation suggests that GPUs should perform insertions on one configuration at a time, dividing insertions among many cores.

The optimal implementation of the post-simulation analysis step is not only a function of the amount of global memory, but also of the speed at which it can be read or written to. Bandwidth between global memory and GPU cores can be over 100 GB/s [12], whereas bandwidth between global memory and CPU memory is typically only 2–6 GB/s [10]. Therefore, reading and writing data between the CPU and GPU could become a computational bottleneck, and ideally should be kept to a minimum in order to take advantage of the two aforementioned strengths of GPUs. One example of data transfer between the CPU and GPU is reading configuration data. Clearly, this data transfer would account for a decreasing share of the computation time if an increasing number of test moves are to be performed per configuration on the GPU. However, as mentioned above, performing more test moves per configuration decreases sampling efficiency. Therefore, an *optimal* number of test moves per configuration must be found in order to maximize performance on GPUs.

Reading configuration data is not the only fixed performance cost per configuration demanded by chemical potential calculations on GPUs. If calculations are performed on a system with electrostatic interactions, then these calculations require the electrostatic potential as a function of particle positions. The potential is typically computed using mesh-based Ewald sums [13], which involve interpolating over a three-dimensional grid that is constant for a given configuration. Ewald sums also involve short-range interactions between pairs of particles. Calculating these interactions efficiently requires a cell list [14, p. 550], a data structure that is constant for a given configuration, just like the long-range electrostatic grid. These two additional fixed performance costs further favor large numbers of insertions per configuration.

To the best of our knowledge, no prior publication has described an implementation of chemical potential calculations on GPUs. As mentioned earlier, GPUs have dramatically improved the performance of other types of calculations involving systems of particles. For example, a molecular dynamics package on GPUs, HOOMD-blue, can speed up simulations by one to two orders of magnitude in comparison to a single CPU core when executed on an NVIDIA GTX 480 graphics card [10]. This card, like other GPUs, is designed specifically for computation with high parallelism and arithmetic intensity. On the other hand, CPUs have a higher fraction of transistors devoted to data caching and flow control. If we can achieve a speed-up for chemical potential calculations similar to what has been observed for other types of calculations involving particles, then dense, strongly-interacting systems will become more accessible. Achieving the maximum speed-up requires locating the optimal number of insertions per configuration that balances sampling efficiency with GPU performance. Accordingly, the goal of this paper is to implement chemical potential calculations on GPUs while at the same time addressing the problem of identifying the optimal number of insertions/deletions.

This paper focuses on three methods for calculating the chemical potential that involve significant amounts of post-simulation analysis: TPI, Bennett's method, and overlap sampling (OS) [6], which is closely related to Bennett's method. Although this paper is concerned primarily with calculating the chemical potential, these methods can be used to calculate other types of free energy differences as well. We summarize the working equations in Section 2. In Section 3 we describe the implementation of these techniques on GPUs. In Section 4 we test our implementation for two systems: water and Lennard-Jones particles, and we also explore the effects of varying the number of insertions per configuration. The main conclusions are summarized in Section 5.

2. Chemical potential methods

The chemical potential methods that we examine involve computing the changes in potential energy upon randomly inserting an additional "test particle" into the N-particle system. These energy changes can be related to the excess chemical potential through the formula [4]

$$\exp(-\beta\mu^{ex}) = \langle \exp(-\beta u)\rangle_N, \tag{1}$$

which is valid for the *NVT* ensemble. In the above equation, $\beta = 1/kT$ (k is Boltzmann's constant), and μ^{ex} is the difference between the chemical potential and that of an ideal gas at the same density and temperature. The angle brackets with the 'N' subscript indicate an ensemble average performed on the N-particle system. u is the change in potential energy for a single random insertion.

In addition to test insertions, test deletions in the $N+1$-particle system are required in the Bennett and OS methods. The OS method relates the potential energy changes from both insertions and deletions to the chemical potential with the formula [6]

$$\exp(-\beta\mu^{ex}) = \frac{\langle \exp(-\beta u/2)\rangle_N}{\langle \exp(\beta u/2)\rangle_{N+1}}, \tag{2}$$

valid for the *NVT* ensemble. u is the change in the potential energy for either a single random insertion in the N-particle system or deletion in the $N+1$-particle system. Bennett's method is based on a similar formula with an added weighting function [6],

$$\exp(-\beta\mu^{ex}) = \frac{\langle w(u)\exp(-\beta u/2)\rangle_N}{\langle w(u)\exp(\beta u/2)\rangle_{N+1}} \tag{3}$$

$$w(u) = 1/\cosh[\beta(u-C)/2] \tag{4}$$

$$\beta C = \beta\mu^{ex} + \ln(n_1/n_0), \tag{5}$$

valid for the *NVT* ensemble. $w(u)$ is chosen to minimize the variance of $\beta\mu^{ex}$ and depends on a parameter C that is not known beforehand. C in turn depends on the number of insertions n_0

and deletions n_1, as well as the chemical potential. Bennett's method assumes that insertions and deletions are uncorrelated, an assumption that becomes increasingly worse with increasing numbers of test moves in the same configuration, as demonstrated in Section 4.

Since C is not known beforehand, it must be solved for self-consistently using Eqs. (3) and (5). One way to do this is by running a series of chemical potential calculations on the same trajectory, iterating over C and μ until Eq. (5) is satisfied. Alternatively, Eq. (3) can be rewritten in terms of the probability density functions of changes in the potential energy due to insertions and deletions, $f(u)$ and $g(u)$ respectively,

$$\exp[-\beta\mu^{ex}] = \frac{\int w(u)\exp(-\beta u/2)f(u)du}{\int w(u)\exp(\beta u/2)g(u)du} \tag{6}$$

$f(u)$ and $g(u)$ can be computed during the course of a single chemical potential calculation by simply creating histograms of insertion or deletion potential energies. Then Eq. (6) can be solved together with (5) using a simple root-finding algorithm like the bisection method.

While Eqs. (1)–(3) are all derived for the *NVT* ensemble, the *NPT* ensemble is usually more convenient for comparison with experiments. The *NPT* analogue of Eq. (1) is well known [15]:

$$\exp(-\mu^{ex}) = \langle\exp(-\beta x)\rangle_N \tag{7}$$

where $x = u - \ln(V/\langle V\rangle)/\beta$. We can obtain similar expressions for the OS method and Bennett's method by performing derivations nearly identical to those in Refs. [6,5], respectively. For the OS method, we obtain

$$\exp(-\beta\mu^{ex}) = \frac{\langle\exp(-\beta x/2)\rangle_N}{\langle\exp(\beta x/2)\rangle_{N+1}}. \tag{8}$$

For Bennett's method, we obtain

$$\exp(-\beta\mu^{ex}) = \frac{\langle w(x)\exp(-\beta x/2)\rangle_N}{\langle w(x)\exp(\beta x/2)\rangle_{N+1}}. \tag{9}$$

Eqs. (1)–(3) or (7)–(9) yield estimates for the chemical potential that can be transformed into fugacities, which are convenient when comparing to experiments, using the following relation:

$$f = \frac{\rho}{\beta}\exp(\beta\mu^{ex}) \tag{10}$$

where ρ is the density of the fluid.

3. Implementation

A convenient way to implement codes on GPUs is in the CUDA C programming environment. In this environment, code to be run on the GPU is organized into functions that are executed multiple times in parallel per call. These functions are referred to as *kernel functions*, and executions within the same call are referred to as *threads*. Threads are executed in batches called *blocks*, the size of which is defined by the user. Depending on their size, 15–120 blocks can be running simultaneously on an NVIDIA GTX 480, for example, though an almost unlimited number can be queued by a single call to a kernel function [16]. Within a block, threads can access a 48 kB pool of shared memory. Threads from any block can access a much larger pool of global memory that can be up to 6 GB. However, transactions with global memory are slower than those with shared memory. Threads within a block can be synchronized at any point in the code, in which case they wait for all other threads within the same block to reach the point of synchronization. In addition, threads can be given exclusive read/write/modify access to a 32-bit or 64-bit word in global memory by using the *atomic* operations provided by CUDA. Synchronization and atomic operations make CUDA flexible enough to implement algorithms that are not strictly parallel, e.g. computing histograms.

We have chosen to code the chemical potential calculation as a single kernel function that is called once per configuration. We could parallelize the calculation at a coarser level, though parallelizing insertions within a configuration appears to be most appropriate for GPUs, as discussed in Section 1. We must still decide how to divide the work among threads within a single call to the kernel function. Two common approaches to this type of problem are the *scatter* and *gather* paradigms [11]. In the *scatter* paradigm, we assign to each thread an input datum, in this case a coordinate of a given particle in the configuration. The thread would then compute all contributions of that input datum to the output data, in this case the energies of all insertions or deletions performed in the box. On the other hand, under the *gather* paradigm, we assign each thread to one or more insertions or deletions, and the thread would gather all contributions from existing particles to the energy of the insertions or deletions. In general, the gather paradigm is preferred, since it avoids atomic operations and barrier synchronization, operations that reduce performance. However, the scatter paradigm can be more attractive if the output data is more *regular* than the input data. In the case of chemical potential calculations, the input data is very irregular; particle coordinates follow a non-linear spatial distribution function. Consequently, the locations of deletions would also be very irregular. We perform insertions at random, though we could equivalently perform them on a regular grid, since the frame of reference of the configurations is arbitrary. Given the regularity of the output data, a scatter implementation of the chemical potential calculation could potentially perform well, however it is outside the scope of this paper. Instead, we have opted for a gather implementation in order to avoid atomic operations and barrier synchronization. In this case there is no benefit to regularizing the output data, so we perform insertions at random using the CURAND library [17] to generate random numbers independently in each thread.

We must still contend with the irregularity of the input data. One established technique for regularizing input data is the combination of truncation of interactions and *binning* [11]. In the context of particle simulations, binning is usually achieved through a cell list [14], which can be computed once per configuration. Fortunately, there already exists a fast implementation of a cell list on GPUs in the molecular dynamics package HOOMD-blue. This package can read in a configuration and output the contents of a given cell, as well as enumerate all neighboring cells in a cubic neighborhood of arbitrary radius. We felt it would be wise to further regularize the input data, since we could potentially be performing very large numbers of insertions per configuration of a dense, strongly-interacting system. The data contained in the cubic neighborhood of cells is still somewhat irregular, since some particles near the corners and edges of the neighborhood will *always* be outside the truncation radius of particles inside the center cell. Therefore, we shave off these corners and edges from the neighborhood. By eliminating extraneous particles in these regions, we improve performance of the kernel function.

Having determined how to divide work among threads, we must decide how to divide threads among blocks. Threads within the same cell must frequently access the coordinates of particles within that cell and its neighborhood. By storing these coordinates in the pools of high-bandwidth shared memory belonging to each block, we might improve performance. Therefore, we assign to each block a cell within the cell list.

Regularizing the coordinates of particles in a configuration accelerates the calculation of short-ranged, pair-wise energies arising from Lennard-Jones interactions and the short-range

portion of Coulombic interactions. For the long-range portion of Coulombic interactions, the relevant input data has already been regularized as part of the PPPM method [18]. The method takes as input the coordinates of partial charges in the system, and outputs the electrostatic potential computed on a regular grid. We then simply interpolate this grid to find the potential at the location of inserted or deleted particles, and multiply by the their partial charges to find the resulting changes in energy. Note that inserted or deleted particles do *not* need to be included in the electrostatic potential itself, since the insertion or deletion is happening in just the central periodic image. For the deletion case, we must additionally subtract a spurious self-interaction term for each particle in the deleted test molecule,

$$U_{\text{self},i} = \frac{2\kappa}{\sqrt{\pi}} q_i^2 + \sum_{j \neq i}^{N_{tp}} \frac{q_i q_j \text{erf}(\kappa |\vec{r}_i - \vec{r}_j|)}{|\vec{r}_i - \vec{r}_j|} \tag{11}$$

since each particle i and its j bonded neighbors already exist inside the configuration. In the above equation, q_i is the partial charge of particle i. κ is reciprocal length parameter describing the Gaussian screening charge density distribution for a given particle i,

$$\rho_{\text{Gauss}}(r) = -q_i(\kappa^2/\pi)^{3/2} \exp(-\kappa^2 r^2). \tag{12}$$

In Eq. (11), $\vec{r}_i$ is the position vector of particle i. N_{tp} is the number of sites within the test molecule being deleted. The self-interaction term in Eq. (11) can be computed ahead of time, and the electrostatic potential grid must be computed only once per configuration. This grid can be obtained from an existing PPPM implementation in HOOMD-blue [19] with some modification, since HOOMD-blue is a molecular dynamics package and therefore requires only the electric field, not the electrostatic potential.

Algorithm 1 Perform insertions/deletions

Require: N_{cell} blocks are run on device
Require: blockDim $\cdot$ N_{cell} CURAND states are initialized
$c \leftarrow$ blockIdx
$I_{c,T} \leftarrow 0$
$I_{c,O} \leftarrow 0$
$D_{c,O} \leftarrow 0$
$I_{th,T} \leftarrow 0$
$I_{th,O} \leftarrow 0$
$D_{th,O} \leftarrow 0$
for $i = 0$ to N_{th} **do**
  $j \leftarrow$ threadIdx $\cdot$ $N_{th} + i$
  if $j < N_{del,c}$ **then**
    $u \leftarrow U_{del}(j)$
    $b \leftarrow B(u, V)$
    atomicAdd($P_{del,b}$, 1)
    $D_{th,O} \leftarrow \exp(\beta u/2)$
  else
    $u \leftarrow U_{ins}()$
    $b \leftarrow B(u, V)$
    atomicAdd($P_{ins,b}$, 1)
    $I_{th,T} \leftarrow \exp(-\beta u)$
    $I_{th,O} \leftarrow \exp(-\beta u/2)$
  end if
end for
atomicAdd($I_{c,T}$, $I_{th,T}$)
atomicAdd($I_{c,O}$, $I_{th,O}$)
atomicAdd($D_{c,O}$, $D_{th,O}$)

The kernel function is summarized in Algorithm 1. This pseudocode describes how the kernel function computes the sums of Boltzmann factors for insertions (I_c) or deletions (D_c) for each block c. These sums are computed for both the TPI method (T) and the OS method (O). These sums are consolidated into estimates for the chemical potential for either the *NVT* or *NPT* ensemble outside of the kernel function on the CPU. Alternatively, the chemical potential for the TPI and OS methods can be estimated using histograms (P_{ins} and P_{del}), which are constructed by the binning function $B(u, V)$, where u is the change in potential energy and V is the volume of the current configuration. Histograms are the sole means of estimating the chemical potential using Bennett's method, and the optimal C value is computed at the end of the simulation trajectory on the CPU.

Algorithm 2 Perform single deletion, $U_{del}(j)$

Require: $\mathbf{X}_c$ is stored in shared memory.
Require: $\mathbf{\Phi}$ is stored in global memory.
$u \leftarrow 0$
for $k < N_{tp}$ **do**
  $\vec{r}_k \leftarrow \vec{x}_{j,k}$
  $u \leftarrow u + U_{sr}(\mathbf{X}_c, \vec{r}_k) + U_{lr}(\mathbf{\Phi}, \vec{r}_k) - U_{self}(k)$
end for
return u

Algorithm 3 Perform single insertion, $U_{ins}(\{\xi\})$

Require: $\mathbf{X}_c$ is stored in shared memory.
Require: $\mathbf{\Phi}$ is stored in global memory.
$u \leftarrow 0$
$q \leftarrow Q(\{\xi\})$
$\vec{r}_{cent} \leftarrow I(\{\xi\})$
for $k < N_{tp}$ **do**
  $\vec{r}_k \leftarrow \vec{r}_{cent} + R(q) \cdot \vec{d}_k$
  $u \leftarrow u + U_{sr}(\mathbf{X}_c, \vec{r}_k) + U_{lr}(\mathbf{\Phi}, \vec{r}_k)$
end for
return u

The kernel function performs a total of $N_{th} \cdot \text{blockDim} \cdot N_{cell}$ insertions and deletions, or perturbations, divided among blockDim$\cdot N_{cell}$ threads. Each thread performs N_{th} perturbations, and during each perturbation, either $U_{del}(j)$ (Algorithm 2) or $U_{ins}(\{\xi\})$ (Algorithm 3) is called to compute the deletion or insertion potential energy, respectively. Both functions read values from arrays containing the coordinates of particles in the cell neighborhood, $\mathbf{X}_c$, and the electrostatic potential grid, $\mathbf{\Phi}$. Additionally, $U_{ins}(\{\xi\})$ takes as input a set of randomly generated numbers $\{\xi\}$ to generate positions for the centers of mass of insertions, $\vec{r}_{cent}$, as well as quaternions [14], q, which then rotate the nominal positions of the N_{tp} sites belonging to the inserted molecule, $\vec{d}_k$, using a rotation matrix, $R(q)$.

4. Validation and performance

4.1. Lennard-Jones particles

Although our code is optimized for dense, strongly-interacting systems with complicated forcefields, we first report results for a simple Lennard-Jones (LJ) liquid at $T^* \equiv kT/\epsilon = 0.851$, a temperature well below the critical temperature ($T_c^* \approx 1.316$ [20]), and $\rho^* \equiv \rho\sigma^3 = 0.8$, a density close to saturation ($\rho_{sat}^*(T^*) \approx 0.77$). Here, σ and ϵ denote the LJ size and energy parameters. LJ systems near these conditions are known to converge easily using all three chemical potential methods that we examined [6].

Each chemical potential method investigated here requires configurations sampled from an ensemble, so we generated these configurations using *NVT* molecular dynamics (MD) in GROMACS 4.5.3 [21]. In principle we could have also generated

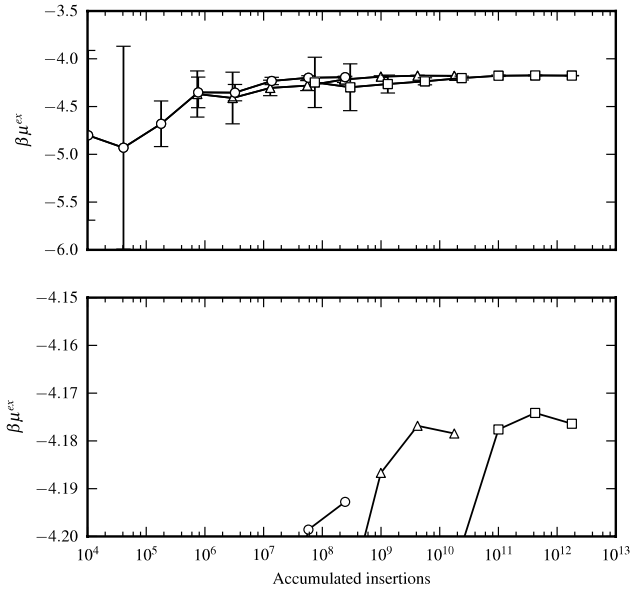

Fig. 1. Converging estimate of the chemical potential using the TPI method for different values of the number of insertions per configuration. ◯, $5.1 \cdot 10^3$ insertions; △, $3.7 \cdot 10^5$ insertions; □, $3.7 \cdot 10^7$ insertions. The system consists of 2880 LJ particles at $T^* = 0.851$ and $\rho^* = 0.8$. The two graphs represent the same data with error bars omitted in the bottom plot for clarity.

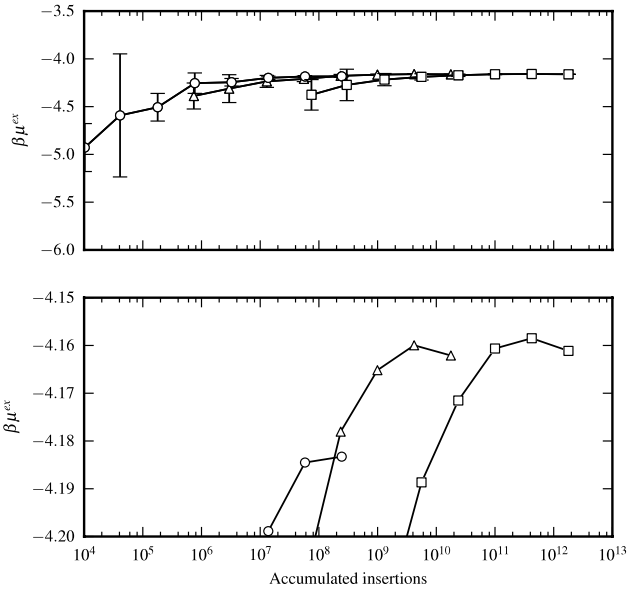

Fig. 2. Converging estimate of the chemical potential using the OS method for different values of the number of insertions per configuration. ◯, $5.1 \cdot 10^3$ insertions; △, $3.7 \cdot 10^5$ insertions; □, $3.7 \cdot 10^7$ insertions. All runs are performed with 2880 deletions per configuration. The system is the same as in Fig. 1.

these configurations on the GPU using Gromacs accelerated with Simtk.org/Open-MM [22]. However, this code does not accelerate simulations of dense, strongly-interacting systems to nearly the same extent as it does simulations of other systems, notably systems with implicit solvent [23]. To the best of our knowledge, this limitation is true for other existing GPU codes as well, and in Section 4.2, we describe calculations involving the TIP4P/2005 water model [24], a system that suffers from this limitation. Therefore, we have chosen to perform molecular dynamics on CPUs in order to optimally allocate available GPU and CPU resources.

For the LJ system, we simulated 2880 particles with cut-off $r^*_{\text{cut}} = 2.5$ and standard long-range corrections [14]. We maintained the temperature using the Nose–Hoover thermostat with $\tau^* \equiv t\sigma(m/\epsilon)^{1/2} = 5.65$ [25], where m is the mass of an LJ particle. We set the time step to $\Delta t^* = 0.005$ and equilibrated the system for $1.8 \cdot 10^6$ time steps. We then output configurations every 2000 time steps, generating a total of 47 807 equilibrium configurations. Strictly speaking, we would have also needed to generate equilibrium configurations from a system of 2881 particles at the same temperature and density, since the OS and Bennett's method are two-stage methods. However, using one set of configurations is a good approximation provided the system is large enough [6].

We performed chemical potential calculations with insertions per configuration varying from $5.1 \cdot 10^3$ to $3.7 \cdot 10^7$ in number. For methods requiring deletions, we held the number of deletions fixed at 2880, the total number of particles in the system. This number is already nearly a factor of two smaller than the lowest number of insertions we examined, so performing deletions consumes a small fraction of the total CPU time, and performing even fewer would provide little benefit. We report the convergence of chemical potential calculations as a function of accumulated insertions into the same MD trajectory in Figs. 1–3. The converged estimate of $\beta\mu^{ex}$ for runs with the highest number of insertions per configuration varies from approximately -4.16 to -4.18, with error bars that are much smaller than the size of the symbols. Error bars are computed using the method of Flyvbjerg [26]. Estimates like these in which the bias is much greater than the variance have been widely observed with these chemical potential methods

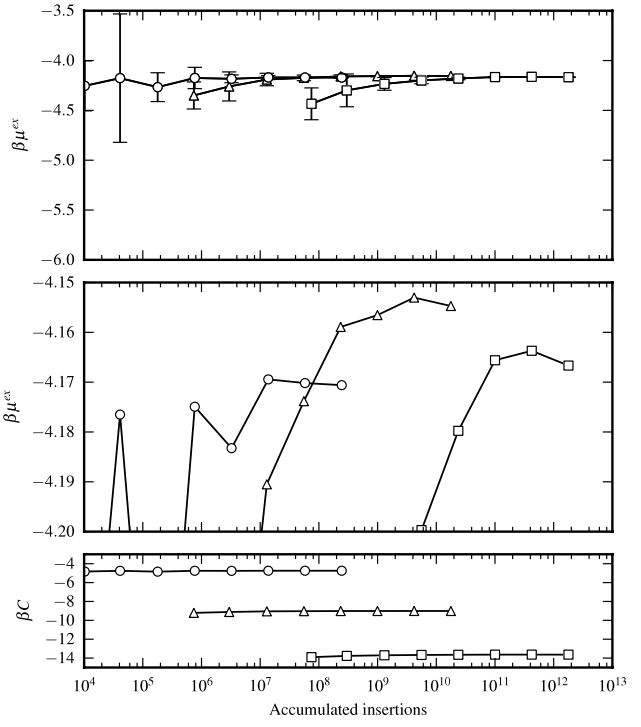

Fig. 3. Converging estimate of the chemical potential using Bennett's Acceptance Ratio method for different values of the number of insertions per configuration. ◯, $5.1 \cdot 10^3$ insertions; △, $3.7 \cdot 10^5$ insertions; □, $3.7 \cdot 10^7$ insertions. All runs are performed with 2880 deletions per configuration. The bottom plot shows the convergence of parameter βC in Bennett's Acceptance Radio method (see Eq. (5) for details). The system is the same as in Fig. 1.

[6,7]. We note that a variation of 0.02 in $\beta\mu^{ex}$ leads only to a 2% change in fugacity. Furthermore, these estimates are all close to -4.12, the value of $\beta\mu^{ex}$ predicted by the equation of state of Johnson et al. [27].

Bennett's method appears to converge faster than the OS method, which in turn converges faster than the TPI method, especially for lower numbers of insertions per configuration.

For example, Bennett's method with $5.1 \cdot 10^3$ insertions per configuration is already within the window of $-4.20 < \beta\mu^{ex} < -4.15$ with slightly less than 10^6 total insertions (Fig. 3), whereas the OS method with the same number of insertions per configuration does not enter this window until 10^7 insertions (Fig. 2), and the TPI method does not reach it until just under 10^8 insertions (Fig. 1). This improvement in performance is close to what was observed in Ref. [6] for an LJ system at $\rho^* = 0.8$ and $T^* = 1.0$ with 1 insertion per configuration. For higher numbers of insertions per configuration, the improvement in performance is more modest, and all three methods appear to eventually exhaust the configuration space of a test particle in a given N-particle configuration. This exhaustion manifests itself as nearly identical convergence trajectories for runs with $3.7 \cdot 10^5$ or more insertions per configuration. Notice in Figs. 1–3 how these trajectories have nearly the same shape, but are merely shifted along the abscissa. Interestingly, when these trajectories are computed using Bennett's method, they are also slightly shifted along the ordinate. We attribute this shifting to inaccuracy in βC. Although βC is well converged, as demonstrated in Fig. 3, it is still computed under the assumption of uncorrelated insertions and deletions in a given N-particle configuration. Clearly, this assumption is invalid when the configuration space for a test particle has been exhausted.

4.2. Water

We report results for a dense, strongly-interacting system of saturated liquid water that is in principle relevant for a wide range of biomolecular, process engineering, pharmaceutical, and other important applications. However, we chose a system at an elevated temperature of 470 K ($T/T_c = 0.73$) in order to make calculations more feasible for less efficient chemical potential methods like TPI. We modeled water with the TIP4P/2005 force field [24], in which the rigid molecule is represented by three massless partial charges and an LJ site. We truncated LJ interactions at 0.9 nm and applied standard long-range corrections [14]. To calculate electrostatic interactions, we used the Smooth Particle Mesh Ewald (SPME) method [28]. We truncated the short-ranged component at 0.9 nm, and used fourth-order interpolation and a maximum grid-spacing of 0.1 nm for the long-ranged component. We set κ at 3.47 nm^{-1}. During the post-simulation chemical potential calculation, we used the PPPM method [18] instead of the SPME method, keeping the short-ranged cut-off at 0.9 nm while setting κ at 5.12 nm^{-1} and the grid-spacing at 0.036 nm, and using sixth-order interpolation.

We approximated a saturated liquid by fixing the pressure at 0 bar. Although this pressure is slightly below the vapor pressure, the system cannot vaporize during the course of the simulation because it still needs to overcome its latent heat of vaporization. To maintain this pressure as well as a temperature of 470 K, we ran *NPT* MD in Gromacs 4.5.3 using the Parrinello–Rahman barostat and the Nose–Hoover thermostat, both with $\tau = 10$ ps [25,29]. We set the time step at 2 fs and equilibrated a system of 2880 molecules for 5 ns. We then stored equilibrium configurations every 1 ps, generating a total of 127 742 configurations.

We performed chemical potential calculations with insertions per configuration varying from $7.1 \cdot 10^3$ to $2.6 \cdot 10^7$ in number. We held the number of deletions fixed at 2880 for methods requiring deletions. We report the convergence of these calculations as a function of accumulated insertions into the same MD trajectory in Figs. 4–6. The OS method and Bennett's method agree well for runs with the highest number of insertions per configuration: they yield estimates of $\beta\mu^{ex}$ that differ by less than 0.005. The TPI method does not converge as satisfactorily, as demonstrated by the three runs in Fig. 4 that give estimates varying approximately from −5.84 to −5.58. The bias in these runs appears to be much larger

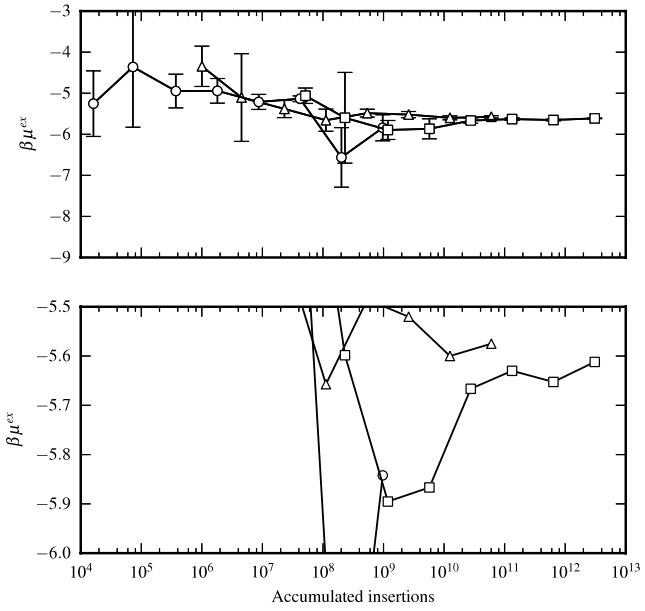

Fig. 4. Converging estimate of the chemical potential using the TPI method for different values of the number of insertions per configuration. ◯, $7.1 \cdot 10^3$ insertions; △, $5.0 \cdot 10^5$ insertions; □, $2.6 \cdot 10^7$ insertions. The system consists of 2880 water molecules at 470 K and 0 bar. The two graphs represent the same data with error bars omitted in the bottom plot for clarity.

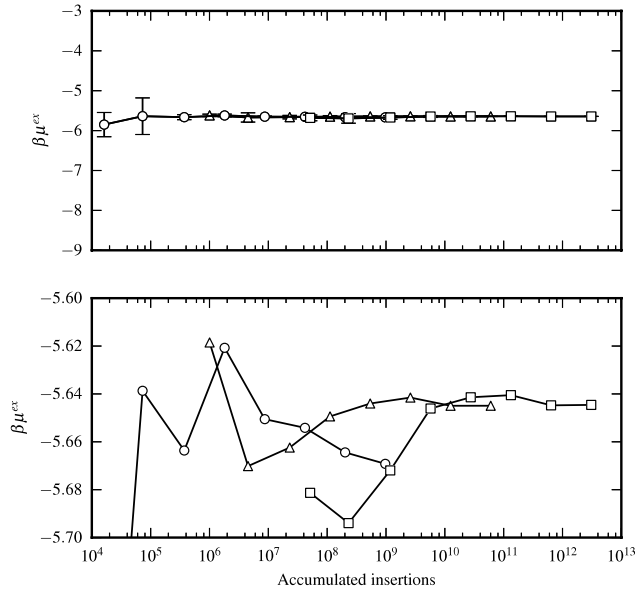

Fig. 5. Converging estimate of the chemical potential using the OS Method for different values of the number of insertions per configuration. ◯, $7.1 \cdot 10^3$ insertions; △, $5.0 \cdot 10^5$ insertions; □, $2.6 \cdot 10^7$ insertions. All runs are performed with 2880 deletions per configuration. The system is the same as in Fig. 4. Note that the ordinate on the bottom plot is zoomed in relative to the bottom plot of Fig. 4.

than the variance, just as in the case of an LJ liquid. Unlike for the LJ liquid, the TPI method does not appear exhaust the configuration space of the test molecule in liquid water. This observation is not surprising given the presence of rotational degrees of freedom in the case of water molecules.

Intriguingly, both the OS and Bennett's methods both appear to exhaust the configuration space of the test molecule at roughly the same point as they do for an LJ liquid: between 10^5 and 10^6 insertions per configuration. Moreover, both of these liquids are near the saturation point, and their values of T/T_c are comparable. This suggests that fluids at similar thermodynamics conditions have similar statistics with respect to insertions and deletions

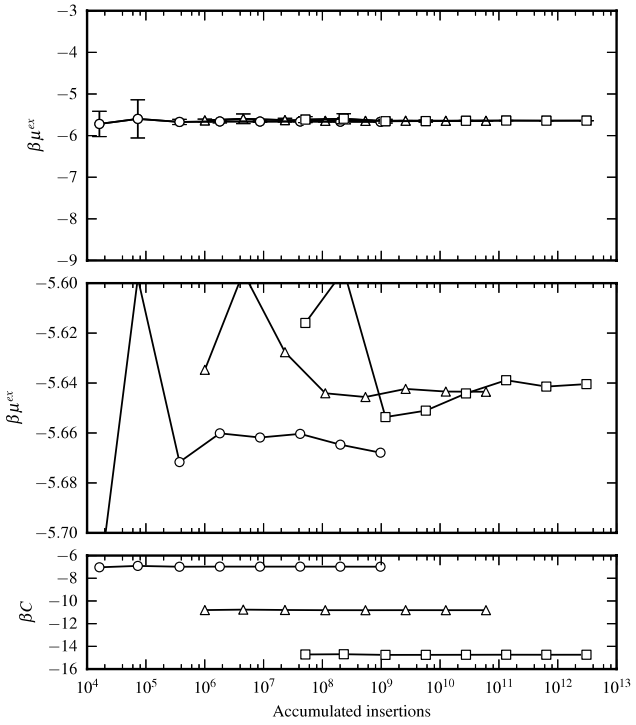

Fig. 6. Converging estimate of the chemical potential using Bennett's Acceptance Ratio method for different values of the number of insertions per configuration. ◯, $7.1 \cdot 10^3$ insertions; △, $5.0 \cdot 10^5$ insertions; □, $2.6 \cdot 10^7$ insertions. All runs are performed with 2880 deletions per configuration. The bottom plot shows the convergence of parameter βC in Bennett's Acceptance Radio method (see Eq. (5) for details). The system is the same as in Fig. 4. Note that the ordinate on the bottom plot is zoomed in relative to the bottom plot of Fig. 4.

if we zero in on the intersection of the *important* configuration space of the N-molecule system and that of the $N + 1$-molecule system. Kofke and coworkers have illustrated the importance of this intersection for free-energy calculations [6]. For an LJ fluid, this intersection is relatively large, so we see the same onset of exhaustion for the TPI method as we do with the other two methods, and little difference in performance. On the other hand, for water the intersection is much smaller, so we observe no onset for the TPI method, and a remarkable improvement in performance using the other two methods, which are designed to target this intersection.

Despite the weaknesses of the TPI method, even its imperfectly converged results are not far off from -5.53, the value of $\beta\mu^{ex}$ corresponding to the vapor pressure calculated by Vega et al. using Gibbs–Duhem integration [30]. Moreover, the maximum variation of $\sim$0.2 in $\beta\mu^{ex}$ at the end of the trajectories in Fig. 4 corresponds to a change of only $\sim$20% in the fugacity. Fortunately, the availability of the OS and Bennett's methods means that we can do much better. The close agreement between the two methods suggest that any residual bias is minimal.

Better sampling efficiency does not necessarily translate to overall efficiency, a measure of performance that additionally accounts for computational cost. To obtain this second measure, we focus on convergence as a function of CPU time, as shown in Fig. 7. This figure shows that the most efficient way to achieve less than 1% deviation from the converged fugacity is to use Bennett's method with $5 \cdot 10^5$ insertions per configuration. The corresponding execution time is a little over 15 min on a single graphics card, demonstrating how feasible these difficult chemical potential calculations become on GPUs. When we set a looser criterion of 10% deviation, we notice a key difference between the TPI method and Bennett's method. The TPI method favors $2.6 \cdot 10^7$ insertions per configuration, whereas Bennett's method favors

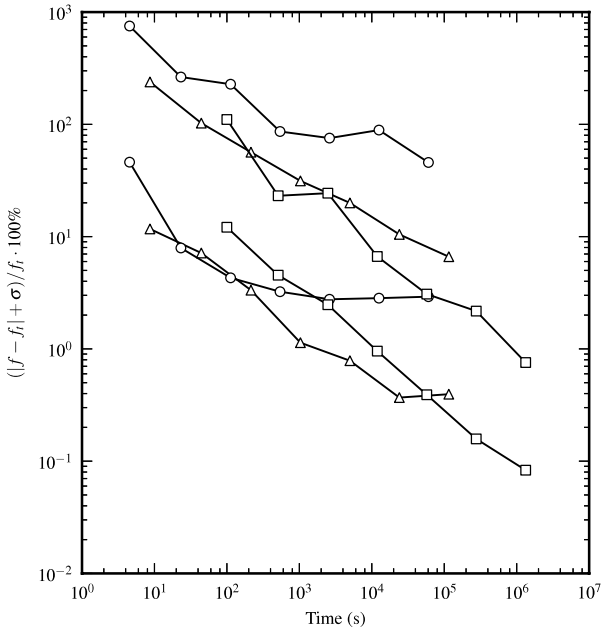

Fig. 7. Converging estimate of the fugacity f as a function of CPU time for the system of Fig. 4. The "true" value of the fugacity is taken to be the value computed at the end of the trajectory during the run with $2.6 \cdot 10^7$ insertions per configuration. σ is the standard deviation using the method of Flyvbjerg [26]. ◯, $7.1 \cdot 10^3$ insertions; △, $5.0 \cdot 10^5$ insertions; □, $2.6 \cdot 10^7$ insertions. Top three curves: the TPI method. Bottom three curves: Bennett's Acceptance Ratio method. All runs are performed on an NVIDIA GTX 580 graphics card.

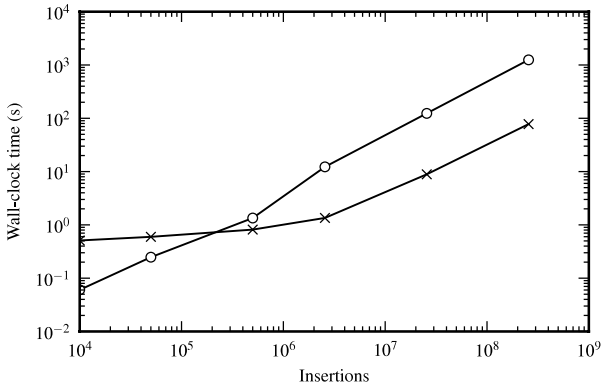

Fig. 8. Time needed to perform a given number of perturbations in a single frame of TIP4P/2005 water at 470 K and 0 bar. ×, system of 2880 water molecules on NVIDIA GTX 580; the number of perturbations equals the number of insertions plus 2880 deletions. ◯, system of 2880 water molecules on 4 × 2.92 GHz CPU cores using GROMACS 4.5.3 [21]. Since GROMACS supports only the TPI method, the number of perturbations is equal to the number of insertions.

$5 \cdot 10^5$ insertions. Yet regardless of the method, lower numbers of insertions per configuration should always give higher sampling efficiencies, implying that higher numbers of insertions per frame are more *computationally* efficient. This implication is confirmed by Fig. 8, which demonstrates that up to $2.6 \cdot 10^6$ insertions per configuration, fixed performance costs overpower the variable cost of insertions.

Interestingly, these fixed costs appear to be specific to GPUs; results for the GROMACS CPU code in Fig. 8 show more or less linear scaling of CPU time all the way down to 10^4 insertions per configuration. These fixed costs also manifest themselves as variations in performance of the GPU code with varying numbers of cells in the cell list, as Fig. 9 shows. The optimal cell list is larger for higher numbers of insertions per configuration. A larger cell list would give a finer approximation for the neighborhood of

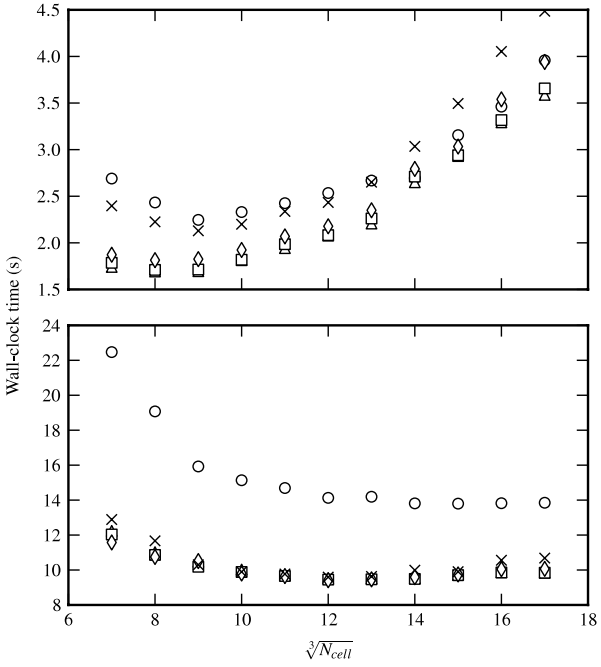

Fig. 9. Performance as a function of the number of insertions per configuration, the number of cells in the cell list, (N_{cell}), and the number of threads in a thread block. ◯, 32 threads; △, 64 threads; □, 128 threads; ◊, 256 threads; ×, 512 threads. Top: time needed to perform $2.6 \cdot 10^6$ insertions in a single frame of 2880 TIP4P/2005 water molecules at 470 K and 0 bar. Bottom: same system with $2.6 \cdot 10^7$ insertions.

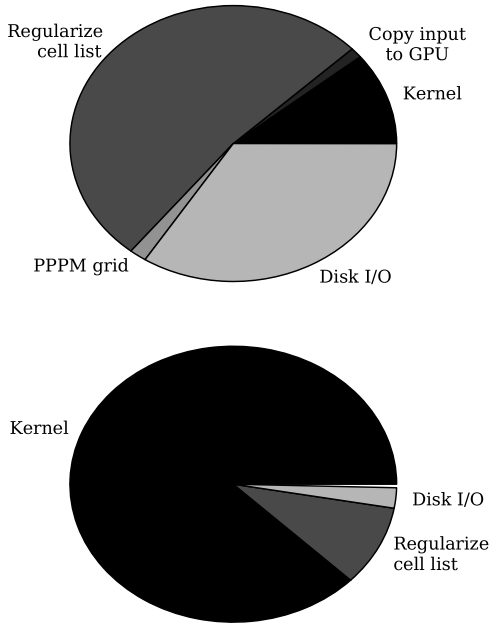

Fig. 10. Fraction of total execution time spent on different tasks while computing chemical potential of a system of 2880 water molecules at 470 K and 0 bar. Top: $4.7 \cdot 10^4$ insertions per configuration. Bottom: $2.6 \cdot 10^7$ insertions per configuration. In both cases the number of deletions per configuration is 2880.

particles that lie within the cut-off of particles in a given cell. This improved approximation would in turn accelerate the calculation of insertion energies, which is a variable cost. On the other hand, a larger cell list would lead to larger fixed costs, since each cell neighborhood has to be regularized by removing extraneous particles near corners and edges, as described in Section 3. Indeed, when we decompose the total CPU time into time spent on specific tasks, as done in Fig. 10, we see that regularizing the cell list as well as other fixed costs dominate the CPU time at lower numbers of insertions per frame. If we operate outside of this regime, we realize the full of benefit of the GPU code, observing a 16-fold reduction in CPU time going from a four-core CPU to a GPU.

5. Conclusions

In this paper we have described an implementation of single-stage and two-stage free-energy methods on GPUs, with emphasis on calculating the chemical potential. We also discussed the optimal number of insertions per configuration that balances sampling efficiency with computational efficiency. This optimum can be $O(10^5)$ for dense, strongly-interacting systems of small molecules, and is higher for a single-stage method than for a two-stage one. Extremely high numbers of insertions per configuration eventually exhaust the configuration space for a test particle, though the onset of this exhaustion can vary by orders of magnitude depending on the system and the method. However, for very different systems at similar reduced conditions, the onset is similar if sampling is restricted to configuration space that is important to both initial and final states corresponding to the free-energy difference.

For $O(10^6)$ insertions per configuration and beyond, the GPU performs over 60 times faster than a single CPU core. This improvement in performance makes GPUs especially well-suited for dense, strongly-interacting systems of small molecules. At low numbers of insertions per configuration, fixed performance costs dominate the execution time. These fixed costs include regularizing the input data, a task that is necessary to improve performance for high numbers of insertions per configuration.

This paper adds to the growing number of calculations on particle systems that have been dramatically accelerated by running on GPUs. Moreover, the code presented in this paper could easily be extended to perform other types of Monte Carlo moves in parallel, such as configurational-bias moves in polymer systems and in dense systems like water. For example, an external Grand-Canonical Monte Carlo code could call functions within our code to perform many trial insertions of a polymer. Each thread would not only insert the first bead, but also grow the rest of the chain using Rosenbluth Sampling [14, pp. 271–287]. Our code would then return the fully-grown polymer configurations and Rosenbluth weights to the client Monte Carlo code, which would choose the permanent insertion configuration according to the weights and a randomly generated number. This approach of packaging GPU code as an external library has already been successfully used in the Simtk.org/OpenMM project to accelerate Molecular Dynamics codes like Gromacs [22].

Acknowledgments

P.G.D. gratefully acknowledges the support of the National Science Foundation (Grant No. CHE-0908265). A.Z.P. would like to acknowledge support for this work from the Department of Energy, Office of Basic Energy Sciences, under grant DE-SC0002128.

References

[1] A. Pohorille, C. Jarzynski, C. Chipot, Good practices in free-energy calculations, The Journal of Physical Chemistry B 114 (2010) 10235–10253.
[2] X. Li, F. Li, Y. Shi, Q. Chen, H. Sun, Predicting water uptake in poly(perfluorosulfonic acids) using force field simulation methods, Physical Chemistry Chemical Physics (2010).
[3] B. Widom, Structure of interfaces from uniformity of the chemical potential, Journal of Statistical Physics 19 (1978) 563–574.
[4] B. Widom, Some topics in the theory of fluids, The Journal of Chemical Physics 39 (1963) 2808.
[5] Charles H. Bennett, Efficient estimation of free energy differences from Monte Carlo data, Journal of Computational Physics 22 (1976) 245–268.

[6] N. Lu, J.K. Singh, D.A. Kofke, Appropriate methods to combine forward and reverse free-energy perturbation averages, The Journal of Chemical Physics 118 (2003) 2977.
[7] M.R. Shirts, V.S. Pande, Comparison of efficiency and bias of free energies computed by exponential averaging, the Bennett acceptance ratio, and thermodynamic integration, The Journal of Chemical Physics 122 (2005) 144107.
[8] A.M. Hahn, H. Then, Measuring the convergence of Monte Carlo free-energy calculations, Physical Review E 81 (2010) 041117.
[9] D.A. Kofke, P.T. Cummings, Quantitative comparison and optimization of methods for evaluating the chemical potential by molecular simulation, Molecular Physics 92 (1997) 973–996.
[10] T.D. Nguyen, C.L. Phillips, J.A. Anderson, S.C. Glotzer, Rigid body constraints realized in massively-parallel molecular dynamics on graphics processing units, Computer Physics Communications 182 (2011) 2307–2313.
[11] W. Hwu, GPU Computing Gems Emerald Edition, Morgan Kaufmann Publishers Inc., 2011.
[12] NVIDIA, CUDA C best practices guide, NVIDIA, Santa Clara, CA, 2012.
[13] M. Deserno, C. Holm, How to mesh up Ewald sums. I. A theoretical and numerical comparison of various particle mesh routines, The Journal of Chemical Physics 109 (1998) 7678–7693.
[14] D. Frenkel, B. Smit, Understanding Molecular Simulation: From Algorithms to Applications, Academic Press, 2002.
[15] K.S. Shing, S.T. Chung, Computer simulation methods for the calculation of solubility in supercritical extraction systems, The Journal of Physical Chemistry 91 (1987) 1674–1681.
[16] NVIDIA, CUDA C programming guide 4.1, NVIDIA, Santa Clara, CA, 2011.
[17] NVIDIA, CUDA toolkit 4.0 CURAND guide, NVIDIA, Santa Clara, CA, 2011.
[18] R. Hockney, J. Eastwood, Computer Simulation Using Particles, Institute of Physics, 1992.
[19] D.N. LeBard, B.G. Levine, P. Mertmann, S.A. Barr, A. Jusufi, S. Sanders, M.L. Klein, A.Z. Panagiotopoulos, Self-assembly of coarse-grained ionic surfactants accelerated by graphics processing units, Soft Matter (2012).
[20] B. Smit, Phase diagrams of Lennard-Jones fluids, The Journal of Chemical Physics 96 (1992) 8639–8640.
[21] B. Hess, C. Kutzner, D. van der Spoel, E. Lindahl, GROMACS 4: algorithms for highly efficient, load-balanced, and scalable molecular simulation, Journal of Chemical Theory and Computation 4 (2008) 435–447.
[22] P. Eastman, V. Pande, OpenMM: a hardware-independent framework for molecular simulations, Computing in Science & Engineering 12 (2010) 34–39.
[23] GPUs—Gromacs, 2012. http://www.gromacs.org/Downloads/Installation_Instructions/GPUs (accessed: 26.04.2012).
[24] J.L.F. Abascal, C. Vega, A general purpose model for the condensed phases of water: TIP4P/2005, The Journal of Chemical Physics 123 (2005) 234505.
[25] S. Nosé, A unified formulation of the constant temperature molecular dynamics methods, The Journal of Chemical Physics 81 (1984) 511.
[26] H. Flyvbjerg, Error estimates on averages of correlated data, Advances in Computer Simulation (1998) 88–103.
[27] J.K. Johnson, J.A. Zollweg, K.E. Gubbins, The Lennard-Jones equation of state revisited, Molecular Physics: An International Journal at the Interface Between Chemistry and Physics 78 (1993) 591.
[28] U. Essmann, L. Perera, M.L. Berkowitz, T. Darden, H. Lee, L.G. Pedersen, A smooth particle mesh Ewald method, Journal of Chemical Physics 103 (1995) 8577–8593.
[29] M. Parrinello, Polymorphic transitions in single crystals: a new molecular dynamics method, Journal of Applied Physics 52 (1981) 7182.
[30] C. Vega, J.L.F. Abascal, I. Nezbeda, Vapor–liquid equilibria from the triple point up to the critical point for the new generation of TIP4P-like models: TIP4P/Ew, TIP4P/2005, and TIP4P/ice, The Journal of Chemical Physics 125 (2006) 034503.