A separable shadow Hamiltonian hybrid Monte Carlo method

Christopher R. Sweet,[1] Scott S. Hampton,[2] Robert D. Skeel,[3] and Jesús A. Izaguirre[1,a)]
[1]*Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, Indiana 46556, USA*
[2]*Oak Ridge National Laboratory, Oak Ridge, Tennessee 37831 USA*
[3]*Department of Computer Science, Purdue University, West Lafayette, Indiana 47907, USA*

(Received 5 May 2009; accepted 2 October 2009; published online 3 November 2009)

Hybrid Monte Carlo (HMC) is a rigorous sampling method that uses molecular dynamics (MD) as a global Monte Carlo move. The acceptance rate of HMC decays exponentially with system size. The shadow hybrid Monte Carlo (SHMC) was previously introduced to reduce this performance degradation by sampling instead from the shadow Hamiltonian defined for MD when using a symplectic integrator. SHMC's performance is limited by the need to generate momenta for the MD step from a nonseparable shadow Hamiltonian. We introduce the separable shadow Hamiltonian hybrid Monte Carlo (S2HMC) method based on a formulation of the leapfrog/Verlet integrator that corresponds to a separable shadow Hamiltonian, which allows efficient generation of momenta. S2HMC gives the acceptance rate of a fourth order integrator at the cost of a second-order integrator. Through numerical experiments we show that S2HMC consistently gives a speedup greater than two over HMC for systems with more than 4000 atoms for the same variance. By comparison, SHMC gave a maximum speedup of only 1.6 over HMC. S2HMC has the additional advantage of not requiring any user parameters beyond those of HMC. S2HMC is available in the program PROTOMOL 2.1. A Python version, adequate for didactic purposes, is also in MDL (http://mdlab.sourceforge.net/s2hmc). © *2009 American Institute of Physics*.
[doi:10.1063/1.3253687]

I. INTRODUCTION

Molecular dynamics (MD) is an important basic sampling method for configuration space of complex systems. It has the ability to make global moves of molecular systems in a straightforward manner. However, MD introduces a systematic time step discretization error into computed averages. Hybrid Monte Carlo (HMC) (Ref. 1) is a rigorous sampling method that employs MD as a global Monte Carlo (MC) move, followed by a Metropolis acceptance criterion. HMC has applications in physics ranging from lattice gauge theory to biomolecular modeling.[2] HMC requires using a time reversible and volume preserving integrator in the MD move to preserve detailed balance.[3] The most popular MD integrator used in HMC is the Verlet or leapfrog integrator. This method is a second-order accurate, time reversible, and symplectic integrator, which provides volume preservation even in the case of nonconservative forces.

Unfortunately, error in the MD step causes the acceptance rate of HMC to decay exponentially for a fixed MD time step with increasing system size N.[4,5] In practice the number of steps scales as $O(N^{1/2})$ because one typically reduces the step size with increasing N. This paper introduces the separable shadow Hamiltonian hybrid Monte Carlo (S2HMC) method. The goal of S2HMC is to improve the acceptance rate of HMC at negligible cost, and thus to benefit from the improved sampling of configuration space to accelerate convergence of averages computed with the method. S2HMC differs from HMC in three ways. First, S2HMC uses a *processed* velocity Verlet (VV) integrator instead of Verlet. The goal of a processing integrator is to increase the effective order of accuracy by using *preprocessing* and *postprocessing* steps.[6,7] The rationale for increasing the effective order of accuracy is that a more accurate integrator has better acceptance rate in HMC. The second difference thus is that S2HMC uses a modified potential energy function, which is conserved to $O(\Delta t^4)$ by the processed method instead of just $O(\Delta t^2)$ by the unprocessed method. The third difference is that S2HMC requires a reweighting step to compensate for modification of the potential energy.

The idea of using a modified potential energy function of higher accuracy and reweighting to improve the acceptance rate of HMC was introduced with the shadow hybrid Monte Carlo (SHMC) method.[8] However, SHMC uses nonseparable Hamiltonians that require relatively expensive generation of momenta and a tuning parameter to balance the cost of rejection of momenta and positions. S2HMC requires no parameters beyond HMC's and is more scalable than SHMC, as numerical results show.

Generalizations of SHMC have been introduced: in particular, the targeted SHMC (TSHMC) (Ref. 9) and the generalized SHMC (GSHMC)[10–12] methods allow momenta to be partially refreshed to improve the probability of accepting momenta from nonseparable Hamiltonians. These protocols also allow more flexibility in the simulation protocol, for instance, by allowing weak stochastic rather than strong stochastic perturbations in the momenta resampling. The approach in S2HMC could be combined with these generalizations.

a)Electronic mail: izaguirr@nd.edu.

The paper proceeds as follows. Section II describes the algorithm so that it can be easily implemented. Sections II and III can be omitted by readers interested in the practical use of the method. Section III gives the theoretical justification. In particular, it is shown that our choice of processed Verlet gives a symplectic and reversible integrator and that S2HMC preserves detailed balance. It is also shown that the modified potential energy is conserved to fourth order. The effect of the reweighting step is also analyzed. Section IV describes the theoretical efficiency of S2HMC. Section V shows numerical tests to illustrate the speedup over HMC. The main reason for speedup is the increased sampling of configuration space due to the higher acceptance rate. Finally, Section VI is a discussion of the results presented.

S2HMC is an elegant method for improving sampling using HMC. It is theoretically justified and preserves detailed balance. It can help overcome bias and energy drift of MD. By using modified potential energies one can get an asymptotic $O(N^{1/4})$ speedup in sampling over HMC.[8] S2HMC can be potentially very helpful in applications that require rigorous sampling for large complex systems.

II. SEPARABLE SHADOW HYBRID MONTE CARLO

We describe HMC and S2HMC with enough detail so that the method can be implemented. A C++ implementation of S2HMC is provided in the program PROTOMOL 2.1.[13] Here we use a Python implementation for easier understanding. The reader interested in implementation details will find them in this section and can safely skip the next two sections. HMC uses a MD step followed by a Metropolis MC step. Let $H(\mathbf{x},\mathbf{p})=\frac{1}{2}\mathbf{p}^{\mathrm{T}}\mathbf{M}^{-1}\mathbf{p}+U(\mathbf{x})$ be the Hamiltonian or total energy of an isolated system. Here $\mathbf{x}$ and $\mathbf{p}$ are vectors with $3N$ elements with all positions and momenta for N atoms, U is the potential energy, and $\frac{1}{2}\mathbf{p}^{\mathrm{T}}\mathbf{M}^{-1}\mathbf{p}$ is the kinetic energy. $\mathbf{M}$ is a diagonal matrix with the mass for each atom repeated three times in the diagonal. This is a separable Hamiltonian and the most common form used in MD. Newton's equations of motion can be written as a Hamiltonian system of ordinary differential equations (ODEs),

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t}=\mathbf{M}^{-1}\mathbf{p},\quad \frac{\mathrm{d}\mathbf{p}}{\mathrm{d}t}=-\nabla_x U(\mathbf{x})\equiv\mathbf{f}(\mathbf{x}), \tag{1}$$

where $\mathbf{f}(\mathbf{x})$, the negative of the gradient of U, is a conservative force. These equations of motion are typically discretized with the VV/leapfrog,[14] which is a second-order accurate, reversible, and symplectic integrator. A step of VV can be written as follows:

(1) Half kick: Let $\mathbf{p}_{n+1/2}=\mathbf{p}_n+(\Delta t/2)\mathbf{f}(\mathbf{x}_n)$,
(2) Drift: Let $\mathbf{x}_{n+1}=\mathbf{x}_n+\Delta t\mathbf{M}^{-1}\mathbf{p}_{n+1/2}$,
(3) Half kick: Let $\mathbf{p}_{n+1}=\mathbf{p}_{n+1/2}+(\Delta t/2)\mathbf{f}(\mathbf{x}_{n+1})$,

where the subscript represents the discretization step and Δt is the time step. To help implement all the methods presented here we describe the code written in Python using the libraries implemented in the Molecular Dynamics Lab (MDL, http://mdlab.sourceforge.net).[15] Enough code is presented to make this discussion concrete. For instance, the half kick step of VV is implemented in line 11 of code listing 1. The code uses Numpy arrays to represent the $3N$ vectors phys.positions, phys.velocities, forces.force, and phys.invmasses, corresponding to $\mathbf{x},\mathbf{p},\mathbf{f}$ and the diagonal matrix $\mathbf{M}^{-1}$, respectively. These are Python arrays optimized for numerical computation, where multiplication is performed elementwise. Note that in the VV implementation, two half kicks are combined into a single step for efficiency in line 18 of code listing 1.

HMC requires using a time reversible and volume preserving integrator in the MD move to preserve detailed balance.[3] We can thus describe HMC using VV as follows. The main HMC function is in code listing 2. HMC calls the Metropolis function (code listing 3) and VelocityVerlet (code listing 1).

Code Listing 1: Velocity Verlet Function

```
def velocityVerlet(phys, forces, steps, timestep, fg):
  # preconditions:
  # phys.positions, phys.velocities, forces.force, phys.invmasses
  #   are numpy arrays with 3 elements for each atom;
  #   these arrays must be set before calling velocityVerlet
  # timestep is the MD timestep in femtoseconds
  # fg has force field data; see below fg.calculateForces
  # postconditions:
  # phys.positions, phys.velocities, and forces are updated
  #   after 'steps' number of MD steps
  phys.velocities+=0.5*timestep*phys.invmasses*forces.force
  phys.positions+=timestep*phys.velocities
  # fg.calculateForces computes forces using phys.positions
  #   and leaves the result in forces.force
  fg.calculateForces(phys, forces)
  step=1
  while (step<=steps):
    phys.velocities+=timestep*phys.invmasses*forces.force
    phys.positions+=timestep*phys.velocities
    fg.calculateForces(phys, forces)
    step=step+1
  phys.velocities+=0.5*timestep*phys.invmasses*forces.force
```

Note that we provide line numbers in parentheses for the codes implementing HMC. We indicate code listing 2 as CL2, etc.

1. Momenta generation: Given initial positions $\mathbf{x}$ generate initial momenta $\mathbf{p}$ from a Maxwell distribution (line 21 in CL2).
2. Compute the initial total energy $H(\mathbf{x},\mathbf{p})$ (line 22 in CL2).
3. MD: Apply several steps of VV, Eqs. (1), (2), (3a), and (3b), starting from $\mathbf{x},\mathbf{p}$ to obtain new positions and velocities $\mathbf{x}'$ and $\mathbf{p}'$ (line 23 in CL2, which calls CL1).
4. Compute the final total energy $H(\mathbf{x}',\mathbf{p}')$ (line 24 in CL2).
5. Metropolis: Accept $\mathbf{x}'$ with probability $\min\{1,\exp((H(\mathbf{x},\mathbf{p})-H(\mathbf{x}',\mathbf{p}'))/(k_{\mathrm{B}}T))\}$ (line 25 in CL2, which calls CL3). If $\mathbf{x}'$ is rejected, keep $\mathbf{x}$ (line 29 in CL2).

Note that HMC performs the Metropolis step based on the difference of the total energy, rather than the potential energy as is customary for methods that sample configuration

space. The reason this step is needed is to eliminate bias introduced by the MD step. HMC is still a method to sample configuration space.

Code Listing 2: Hybrid Monte Carlo

```
def hmc(phys, forces, hmc_steps, md_steps, md_timestep, fg):
  # preconditions:
  # phys.positions (numpy array) must be set
  # phys.temperature (scalar) must be set in Kelvin
  # hmc_steps is number of hmc steps
  # md_steps is number of MD steps per hmc step
  # md_timestep is MD timestep in femtoseconds
  # fg has force field data, and method fg.calculateForces
  step=1
  temp=phys.temperature
  # fg.calculateForces computes forces using phys.positions
  #  & forces are stored in numpy array forces.force
  #  & total energy is stored in scalar forces.totalEnergy
  fg.calculateForces(phys, forces)
  while (step<=hmc_steps):
    # save state in case of rejection:
    # deepcopy(src, dest) deep copies 2 numpy arrays
    deepcopy(phys.positions, currPos)
    # Generate Momenta: Gaussian distribution centered at T
    #  & phys.velocities are updated; uses phys.invmasses
    phys.randomVelocity(temp)
    currE=forces.energies.totalEnergy(phys)
    velocityVerlet(phys, forces, md_steps, md_timestep, fg)
    newE=forces.energies.totalEnergy(phys)
    accept=metropolis(newE,currE,temp)
    if(accept):
      deepcopy(phys.positions, currPos) # save state
    else:
      deepcopy(currPos, phys.positions) # restore state
```

The goal of S2HMC is to use a processed VV method whose shadow Hamiltonian is separable and fourth order,

$$\tilde{\mathcal{H}}(\mathbf{x},\mathbf{p}) = \frac{1}{2}\mathbf{p}^{\mathrm{T}}\mathbf{M}^{-1}\mathbf{p} + U(\mathbf{x}) + \frac{\delta t^2}{24}\mathbf{U}_{\mathbf{x}}^{\mathrm{T}}\mathbf{M}^{-1}\mathbf{U}_{\mathbf{x}} + \mathcal{O}(\delta t^4). \tag{2}$$

Code Listing 3: Metropolis Function

```
import Constants          # for Boltzmann constant
from random import *      # for random()
from math import *        # for exp()

def metropolis(new, curr, temp):
  # new and curr are energies in kcal/mol
  # temp is temperature in Kelvin
  deltaE=new-curr
  if(deltaE<0):
    return 1
  acceptProb=exp(-deltaE/(Constants.boltzmann( )*temp))
  randNum=random( )
  if(randNum<acceptProb):
    return 1
  else:
    return 0
```

The processed Verlet is achieved by applying a preprocessing step to the phase space variables $\mathbf{x}$ and $\mathbf{p}$ prior to the VV propagation and applying the inverse mapping afterward for postprocessing. The theoretical justification of the shadow Hamiltonian, the preprocessing, and postprocessing steps are found in Sec. III.

The preprocessing step is given by

$$\hat{\mathbf{p}} = \mathbf{p} - \frac{\delta t}{24}(\mathbf{U}_{\mathbf{x}}(\mathbf{x} + \delta t\mathbf{M}^{-1}\hat{\mathbf{p}}) - \mathbf{U}_{\mathbf{x}}(\mathbf{x} - \delta t\mathbf{M}^{-1}\hat{\mathbf{p}})), \tag{3a}$$

$$\hat{\mathbf{x}} = \mathbf{x} + \frac{\delta t^2}{24}\mathbf{M}^{-1}(\mathbf{U}_{\mathbf{x}}(\mathbf{x} + \delta t\mathbf{M}^{-1}\hat{\mathbf{p}}) + \mathbf{U}_{\mathbf{x}}(\mathbf{x} - \delta t\mathbf{M}^{-1}\hat{\mathbf{p}})), \tag{3b}$$

requiring an iterative solution for $\hat{\mathbf{p}}$ and a direct computation for $\hat{\mathbf{x}}$.

The postprocessing step is given by

$$\mathbf{x} = \hat{\mathbf{x}} - \frac{\delta t^2}{24}\mathbf{M}^{-1}(\mathbf{U}_{\mathbf{x}}(\mathbf{x} + \delta t\mathbf{M}^{-1}\hat{\mathbf{p}}) + \mathbf{U}_{\mathbf{x}}(\mathbf{x} - \delta t\mathbf{M}^{-1}\hat{\mathbf{p}})), \tag{4a}$$

$$\mathbf{p} = \hat{\mathbf{p}} + \frac{\delta t}{24}(\mathbf{U}_{\mathbf{x}}(\mathbf{x} + \delta t\mathbf{M}^{-1}\hat{\mathbf{p}}) - \mathbf{U}_{\mathbf{x}}(\mathbf{x} - \delta t\mathbf{M}^{-1}\hat{\mathbf{p}})), \tag{4b}$$

requiring an iterative solution for $\mathbf{x}$ and a direct computation for $\hat{\mathbf{p}}$. Since the differences $\mathbf{p}-\hat{\mathbf{p}}$ and $\mathbf{x}-\hat{\mathbf{x}}$ are $\mathcal{O}(\delta t^2)$, a fixed point iteration converges rapidly for both preprocessing and postprocessing.

An implementation in Python of the shadow Hamiltonian computation is shown in code listing 4. This code is used to implement the S2HMC method in code listing 5. The S2HMC method using VV then becomes the following:

(1) Momenta generation: Given initial positions $\mathbf{x}$ generate initial momenta $\mathbf{p}$ from a Maxwell distribution (line 17 in CL5).
(2) Compute the initial shadow energy $\tilde{H}(\mathbf{x},\mathbf{p})$ using Eq. (2). Note that this computation is dependent only on the total energy, the time step, the forces, and the inverse masses (line 20 in CL5, which calls CL4).
(3) Preprocessing: Starting from $\mathbf{x}$ and $\mathbf{p}$, solve iteratively for $\hat{\mathbf{p}}$ and directly compute $\hat{\mathbf{x}}$ using Eqs. (3a) and (3b) (line 23 in CL5). The MD step is done using these processed positions and momenta.
(4) MD: Apply several steps of VV starting from $\hat{\mathbf{x}}$ and $\hat{\mathbf{p}}$ to obtain new positions and velocities, $\hat{\mathbf{x}}'$ and $\hat{\mathbf{p}}'$ (line 25 in CL5, which calls CL1).
(5) Postprocessing: Starting from $\hat{\mathbf{x}}'$ and $\hat{\mathbf{p}}'$, solve iteratively for $\mathbf{x}'$ and directly compute $\mathbf{p}'$ using Eqs. (4a) and (4b) (line 27 in CL5).
(6) Compute the final shadow energy $\tilde{H}(\mathbf{x}',\mathbf{p}')$ using Eq. (2) (line 29 in CL5, which calls CL4).
(7) Metropolis: Accept $\mathbf{x}'$ with probability $\min\{1,\exp((\tilde{H}(\mathbf{x},\mathbf{p})-\tilde{H}(\mathbf{x}',\mathbf{p}'))/(k_{\mathrm{B}}T))\}$. If $\mathbf{x}'$ is rejected, keep $\mathbf{x}$ (lines 30–34 in CL5, which call CL3).
(8) Compute weight: To compute the averages of a quantity $A(\mathbf{x})$ using S2HMC, reweighting of the sequence of A is needed. At each step i of S2HMC, the weight for the canonical ensemble can be computed as

$$\omega_i = \exp((\tilde{H}(\mathbf{x}_i,\mathbf{p}_i) - H(\mathbf{x}_i,\mathbf{p}_i))/(k_{\mathrm{B}}T)), \tag{5}$$

where $\mathbf{x}_i$ and $\mathbf{p}_i$ are the positions and momenta accepted in the Metropolis step. These weights can be used after

a simulation is finished: the reweighted estimator $\tilde{\theta}$ of an average for $A(\mathbf{x})$ is

$$\tilde{\theta} = \frac{\Sigma_i \omega_i A_i(\mathbf{x}_i)}{\Sigma_i \omega_i}. \tag{6}$$

Code Listing 4: S2HMC Shadow Hamiltonian Computation

```
def computeShadow(phys, forces, dt):
  # preconditions:
  # forces.force and forces.totalEnergy have been computed
  # phys.invmasses has been populated
  # dt is MD timestep in femtoseconds
  # postcondition:
  # return shadow: S(x,p)=H(x,p)+dt^2/24U_x^T M^-1 U_x
  dt2d24=dt*dt/24.0
  result=forces.energies.totalEnergy(phys)
  # use numpy function dot for computing dot product
  Uq_M_Uq=dot(forces.force,phys.invmasses*forces.force)
  result+=dt2d24*Uq_M_Uq
  return result
```

III. THEORETICAL JUSTIFICATION

Recall that HMC requires the MD integrator to be volume preserving and reversible in order to satisfy detailed balance. In this section we show that S2HMC satisfies detailed balance by showing the processed leapfrog is symplectic and reversible. We also show that the shadow Hamiltonian $\tilde{H}$ of S2HMC is conserved to fourth order for processed leapfrog. Finally, we derive and analyze the effect of reweighting on statistical efficiency.

A. S2HMC satisfies detailed balance

The goal of S2HMC is to use a processed[7] leapfrog method whose fourth order shadow Hamiltonian is separable,

$$\tilde{\mathcal{H}}(\mathbf{x},\mathbf{p}) = \frac{1}{2}\mathbf{p}^{\mathrm{T}}\mathbf{M}^{-1}\mathbf{p} + \tilde{U}(\mathbf{x}) + \mathcal{O}(\delta t^4), \tag{7}$$

for a modified potential energy $\tilde{U}$. The modified or shadow Hamiltonian is a result of applying backward error analysis to numerical integrators.[7] In the analysis of numerical integrators for Hamiltonian systems, shadow Hamiltonians are quantities that are better conserved than the true Hamiltonian. In particular, a fourth order shadow Hamiltonian is conserved within $O(\Delta t^4)$, where Δt is the discretization time step. For symplectic integrators one can construct shadow Hamiltonians of arbitrarily high order.[16,17]

Code Listing 5: Separable Shadow Hamiltonian Hybrid Monte Carlo

```
def s2hmc(phys, forces, hmc_steps, md_steps, md_timestep, fg):
  # preconditions:
  # phys.positions (numpy array) must be set
  # phys.temperature (scalar) must be set in Kelvin
  # hmc_steps is number of hmc steps
  # md_steps is number of MD steps per hmc step
  # md_timestep is MD timestep in femtoseconds
  # fg has force field data, and method fg.calculateForces
  step=1
  temp=phys.temperature
  # save state in case of rejection:
  # deepcopy(src, dest) deep copies 2 numpy arrays
  deepcopy(phys.positions, currPos)
while (step<=hmc_steps):
  # Generate Momenta: Gaussian distribution centered at T
  # & phys.velocities are updated; uses phys.invmasses
  phys.randomVelocity(T)
  # Compute shadow: S(x,p)=H(x,p)+dt^2/24U_x^T M^-1 U_x
  fg.calculateForces(phys, forces)
  currE=computeShadow(phys,forces,md_timestep)
  # Preprocessing: Iteratively solve for hat_p; compute hat_x
  # These are stored in hat.velocities and hat.positions:
  hat=preprocess(phys)
  fg.calculateForces(hat, forces)
  velocityVerlet(hat, forces, fg, md_steps, md_timestep)
  # Postprocessing: Iteratively solve for x; compute p:
  phys=postprocess(hat)
  fg.calculateForces(phys, forces)
  newE=computeShadow(phys,forces,md_timestep)
  accept=metropolis(newE,currE,temp)
  if(accept):
    deepcopy(phys.positions, currPos) # save state
  else:
    deepcopy(currPos, phys.positions) # restore state
```

A processed integrator is a new hopefully better integrator constructed by doing a preprocessing or change in phase space variables prior to propagation by another integrator. A postprocessor, which is an inverse of the preprocessing mapping, is evaluated when output is required. Processing is advantageous if the processed method is more accurate than the original integrator.[18] In our application the cost of processing is amortized over a number of MD steps performed before doing a Metropolis MC evaluation.

Equation (7) for S2HMC is obtained by applying a preprocessing map to the phase space variables $\mathbf{x}$ and $\mathbf{p}$ prior to the MD propagation and applying the inverse mapping afterward for postprocessing. The map should commute with reversal of momenta and should preserve phase space volume so that the resulting S2HMC ensures detailed balance. We can get a symplectic map using a generating function of the third kind,

$$S(\mathbf{x},\hat{\mathbf{p}}) = \mathbf{x}^{\mathrm{T}}\hat{\mathbf{p}} + \frac{\delta t}{24}(U(\mathbf{x} + \delta t\mathbf{M}^{-1}\hat{\mathbf{p}}) - U(\mathbf{x} - \delta t\mathbf{M}^{-1}\hat{\mathbf{p}})). \tag{8}$$

The map $(\hat{\mathbf{x}},\hat{\mathbf{p}}) = \chi(\mathbf{x},\mathbf{p})$ is given by

$$\hat{\mathbf{x}} = \frac{\partial S}{\partial \hat{\mathbf{p}}}, \quad \mathbf{p} = \frac{\partial S}{\partial \mathbf{x}}. \tag{9}$$

The map $(\hat{\mathbf{x}},\hat{\mathbf{p}}) = \chi(\mathbf{x},\mathbf{p})$ from Eq. (8) is given precisely by Eqs. (3a) and (3b). This map is the "preprocessing" step of S2HMC. In other words, it is the canonical change in variables that preserves the symplectic property of the processed leapfrog. The inverse mapping χ^{-1} is given by Eqs. (4a) and (4b). The inverse mapping is the "postprocessing" step of S2HMC.

The reversibility of the processed leapfrog can be shown

using $\chi(\mathbf{x},-\mathbf{p})=\mathrm{diag}(I,-I)\chi(\mathbf{x},\mathbf{p})$. Thus, since the processed leapfrog is both symplectic and reversible, S2HMC preserves detailed balance.

B. A fourth order shadow Hamiltonian for the processed leapfrog

A fourth order modified or shadow Hamiltonian for the processed leapfrog can be shown to be given by Eq. (2). The numerical solution of a leapfrog is known to be near the analytical solution of a nearby Hamiltonian system whose shadow Hamiltonian is[19]

$$\tilde{\mathcal{H}}_0(\hat{\mathbf{x}},\hat{\mathbf{p}}) = U(\hat{\mathbf{x}}) + K(\hat{\mathbf{p}}) + \tfrac{1}{12}\delta t^2 \mathbf{K}_\mathbf{p}^\mathrm{T}\mathbf{U}_{\mathbf{xx}}\mathbf{K}_p - \tfrac{1}{24}\delta t^2 \mathbf{U}_\mathbf{x}^\mathrm{T}\mathbf{K}_{\mathbf{pp}}\mathbf{U}_\mathbf{x} + \mathcal{O}(\delta t^4). \quad (10)$$

Recall that $K(\mathbf{p})$ is the kinetic energy $\frac{1}{2}\mathbf{p}^\mathrm{T}\mathbf{M}^{-1}\mathbf{p}$.

The transformed variables $\hat{\mathbf{x}}$ and $\hat{\mathbf{p}}$ have as their Hamiltonian $\tilde{H}_0$ given by Eq. (10). The original variables $\mathbf{x}$ and $\mathbf{p}$ are related to the transformed variables by $(\hat{\mathbf{x}},\hat{\mathbf{p}})=\chi(\mathbf{x},\mathbf{p})$, hence $\tilde{H}(\mathbf{x},\mathbf{p})=\tilde{H}_0(\chi(\mathbf{x},\mathbf{p}))$. This is true because the transformation is symplectic.

We find power expansions for $\hat{\mathbf{x}}$ and $\hat{\mathbf{p}}$ of the appropriate order,

$$\hat{\mathbf{x}} = \mathbf{x} + \frac{\delta^2}{12}\mathbf{M}^{-1}\mathbf{U}_\mathbf{x} + \mathcal{O}(\delta^4), \quad (11)$$

$$\hat{\mathbf{p}} = \mathbf{p} - \frac{\delta^2}{12}\mathbf{U}_\mathbf{x}\mathbf{x}\mathbf{M}^{-1}\mathbf{p} + \mathcal{O}(\delta^4). \quad (12)$$

By substituting Eqs. (11) and (12) into Eq. (10), and then finding a power expansion for each term, we obtain the shadow Hamiltonian of Eq. (2).

C. Reweighting in S2HMC

HMC can be thought to be a sampling from a probability density function (PDF) $\rho(\mathbf{x},\mathbf{p})$, which typically is the canonical distribution, and is induced by the Hamiltonian or total energy. Since the PDF for the SHMC method is induced by the shadow Hamiltonian, observables must be reweighted to obtain the values that would have resulted from using the correct PDF. Given that the PDF for the shadow Hamiltonian is $\tilde{\rho}(\mathbf{x},\mathbf{p})$ and the required PDF is $\rho(\mathbf{x},\mathbf{p})$, then for an observable $A(\mathbf{x},\mathbf{p})$ the reweighted estimator $\tilde{\theta}$ for the expected value of A, given a sequence $\{A_i\}$, is given by Eq. (6), where $\omega_i \propto \rho(\mathbf{x}_i,\mathbf{p}_i)/\tilde{\rho}(\mathbf{x}_i,\mathbf{p}_i)$. For the canonical ensemble the weights are given by Eq. (5), noting that the constant of proportionality, equal to the ratio of the partition functions of $\tilde{\mathcal{H}}$ and $\mathcal{H}$, cancels in Eq. (6).

An estimate of the variance of the observable itself can be obtained from[20]

$$\mathrm{Var}(A)\mathrm{estimator} = \frac{\Sigma_i\omega_i}{(\Sigma_i\omega_i)^2 - \Sigma_i\omega_i^2}\sum_i \omega_i(A_i - \tilde{\theta})^2 \quad (13)$$

and is verified in the tests in Sec. V, Table I.

Reweighting of observables leads to an increase in variance of the estimator of the expected value of A. In the following discussion we assume that the observables $A_i = A(\mathbf{x}_i,\mathbf{p}_i)$ are independent. For nonreweighted data, denoting the estimator of the expected value of an observable $A(\mathbf{x},\mathbf{p})$ as θ, then

$$\mathrm{Var}(\theta) = \mathrm{Var}\left(\frac{\Sigma_i A_i}{n}\right) = \frac{\mathrm{Var}(A)}{n}, \quad (14)$$

where n is the number of samples.[21] For the reweighted observable the variance of the estimator of the expected value $\tilde{\theta}$ is

$$\mathrm{Var}(\tilde{\theta}) = \mathrm{Var}\left(\frac{\Sigma_i^n \omega_i A_i}{\Sigma_i^n \omega_i}\right) = \mathrm{Var}(A)\frac{\Sigma_i^n \omega_i^2}{(\Sigma_i^n \omega_i)^2}. \quad (15)$$

Since $(\Sigma_i^n\omega_i^2/\Sigma_i^n\omega_i)^2 \geq 1/n$, the variance of the reweighted estimator of the expected value is greater than or equal to the variance of the nonreweighted observables.

The implication of this observation is that although S2HMC increases the probability of acceptance of the MD step, this improvement needs to be offset by the slower convergence to the expected values of the observables. However, on a finite time scale, the issue is one of adequately exploring phase space rather than the increase in variance of the estimator due to nonuniform weighting. Thus, the ability of S2HMC to sample more efficiently, due to its use of a fourth order processed leapfrog rather than a second-order leapfrog, is the key to the performance benefits of using the method. We show how this happens in Sec. IV.

TABLE I. Mean estimate (in kcal mol^{-1}) and SD estimate of the potential energy for HMC, SHMC, and S2HMC on a 4002 atom water box with periodic boundary conditions. These simulations have 200 000 MD steps using time step of 1 fs and 200 MC steps. The bracketed value for S2HMC is the nonreweighted SD. The reference value was computed using six different initial conditions and running 25 different simulations with different random seeds for the momenta generation (see text).

Method	Mean estimate	SD estimate
Reference	−14 223	15
HMC	−14 259	76
SHMC	−14 238	70
S2HMC	−14 216	73 (64)

IV. THEORETICAL EFFICIENCY

The S2HMC MD move is accepted with probability $\min\{1,\exp(-\beta\Delta\tilde{\mathcal{H}})\}$ where $\Delta\tilde{\mathcal{H}}=\tilde{\mathcal{H}}(\mathbf{x}',\mathbf{p}')-\tilde{\mathcal{H}}(\mathbf{x},\mathbf{p})$. For large systems, we expect that different parts of the system are sufficiently decoupled to behave like independent random variables, leading to $\Delta\tilde{\mathcal{H}}$ being normally distributed. Assuming this to be the case and that $\Delta\tilde{\mathcal{H}}$ has mean $\mu_{\Delta\tilde{\mathcal{H}}}$ and variance $\sigma^2_{\Delta\tilde{\mathcal{H}}}$, then the average acceptance ratio R_S is

$$R_S = \frac{1}{\sqrt{2\pi}\sigma_{\Delta\tilde{\mathcal{H}}}} \times \int_{-\infty}^{\infty} \min\{1, \exp(-\beta x)\} \exp\left(\frac{-(x-\mu_{\Delta\tilde{\mathcal{H}}})^2}{2\sigma_{\Delta\tilde{\mathcal{H}}}^2}\right) dx. \tag{16}$$

The moment distribution function for normal distributions is the expected value of $\exp(tX)$ given by $\mathrm{E}[\exp(tX)]=\exp(\mu t+\sigma^2 t^2/2)$. The cumulant generating function is the logarithm of the moment generating function $\mu t+\sigma^2 t^2/2$. From the volume preservation property of the numerical integrator,[8] we have $\langle \exp(-\beta\Delta\tilde{\mathcal{H}})\rangle_{\tilde{p}}=1$. This is equivalent to saying that the moment-generating function for the random variable $\Delta\tilde{\mathcal{H}}$ evaluated at $-\beta$ is equal to 1 and can be expanded into cumulants $k_1, k_2, \ldots$. Since $\Delta\tilde{\mathcal{H}}$ is expected to be normally distributed, we have $k_1=\mu_{\Delta\tilde{\mathcal{H}}}$, $k_2=\sigma^2_{\Delta\tilde{\mathcal{H}}}$, and $k_n=0\,\forall\, n>2$. Evaluating the cumulant generating function at $-\beta$ we get $\mu_{\Delta\tilde{\mathcal{H}}}=(\beta/2)\sigma^2_{\Delta\tilde{\mathcal{H}}}$. From this and Eq. (16), the average probability of accepting the move is

$$R_S = \mathrm{erfc}\left(\frac{1}{2\sqrt{2}}\beta\sigma_{\Delta\tilde{\mathcal{H}}}\right). \tag{17}$$

Again, for sufficiently large systems we would expect that $\sigma^2_{\Delta\tilde{\mathcal{H}}} \propto N$, where N is the number of atoms.

For higher order shadow Hamiltonians, the improvement in sampling efficiency of SHMC was shown to be asymptotically $O(N^{1/4})$.[8] Thus, using the procedure outlined here for S2HMC to construct higher order Hamiltonians would produce even more efficient methods.

V. NUMERICAL TESTS

We present numerical tests on different size water boxes to illustrate S2HMC's scalability with system size. Tests were performed with the open-source simulator PROTOMOL,[13] which uses the CHARMM 22 force field.[22,23] We tested the method using five boxes of flexible TIP3P (Ref. 24) water with 1002, 2001, 4002, 8001, and 16 002 total atoms. The choice of water boxes reflects our need to test the scalability of hybrid methods on system size and have consistency across models. The simulations were done with periodic boundary conditions at a temperature of 300 K. For HMC, S2HMC, and SHMC, a 100 fs MD trajectory was performed for each of 500 MC steps.

The analysis for the average acceptance probability for the HMC method is identical to that given for S2HMC, except for the substitution of the variance of the original Hamiltonian difference $\sigma^2_{\Delta\mathcal{H}}$. It is evident that the efficiency of the S2HMC method will scale favorably with time step δt when compared with HMC as the standard deviation (SD) $\sigma_{\Delta\tilde{\mathcal{H}}}$ scales with δt^4 instead of δt^2 for $\sigma_{\Delta\mathcal{H}}$. To illustrate this, we assume $\Delta\mathcal{H}$ to have variance σ_a^2 for time step τ_a and $\Delta\tilde{\mathcal{H}}$ to have variance σ_b^2 at time step τ_b. If the MD time interval is of fixed length T then the efficiency is inversely proportional to the time step. The increase in performance due to the S2HMC method S will then be the ratio of the maximum efficiencies for both methods,

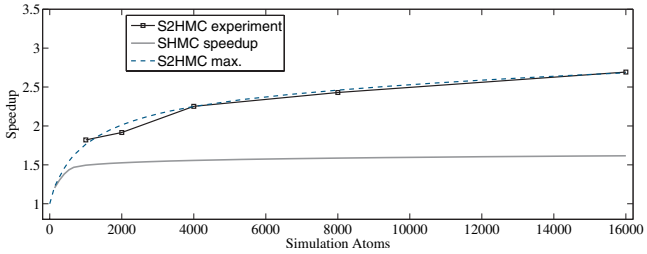

FIG. 1. The increase in performance due to the S2HMC and SHMC methods (speedup). The S2HMC max curve is the analytical solution given in Eq. (18) after determining the variance of the Hamiltonian and shadow Hamiltonian from simulation.

$$S = \frac{\max_{\delta t_b}\left\{\mathrm{erfc}\left(\frac{1}{2\sqrt{2}}\beta\sigma_b\left(\frac{\delta t_b}{\tau_b}\right)^4\right)\delta t_b\right\}}{\max_{\delta t_a}\left\{\mathrm{erfc}\left(\frac{1}{2\sqrt{2}}\beta\sigma_a\left(\frac{\delta t_a}{\tau_a}\right)^2\right)\delta t_a\right\}}, \tag{18}$$

where σ_a, σ_b, τ_a, and τ_b can be measured from simulation.

In Fig. 1, S is shown for different system sizes and can be compared with the performance of SHMC, which is also plotted. It was possible to use time steps of 1 fs for all of the models without the average acceptance probabilities falling below 0.85. The S2HMC maximum curve is the analytical solution given in Eq. (18) after determining the variance of $\Delta\mathcal{H}$ and $\Delta\tilde{\mathcal{H}}$ from simulation. It is clear that there is a significant improvement for all of the boxes of water and an efficiency increase for S2HMC of greater than two times for boxes containing more than 4002 atoms. Equations (3a) and (4b) were solved iteratively until a user-specified convergence was achieved. The cost of these preprocessing and postprocessing steps for S2HMC is negligible and could not be distinguished from statistical error for the runtime across different runs of our tests. As an illustration, the norm-squared tolerance and the number of iterations to converge $\hat{\mathbf{x}}$ and $\hat{\mathbf{p}}$ are given below. The results are the same for water boxes from 1000 to 16 000 atoms within one iteration.

Tolerance	$\hat{\mathbf{x}}$ iterations	$\hat{\mathbf{p}}$ iterations
$1\times10-8$	4	3
$1\times10-12$	5	4
$1\times10-16$	6	5

The estimated variance of the potential energy for the 4002 atom water box is shown in Table I for HMC, SHMC, and S2HMC. This indicates that the correct variance is obtained with the reweighting. The convergence of the average potential energy to its expected values θ and $\tilde{\theta}$ for the 4002 atom water model was measured for HMC and S2HMC. This is shown in Fig. 2 where the error bars represent 1 SD, as estimated by Eq. (14) or Eqs. (15) and (13). This confirms that adequately sampling phase space is more important than the increase in variance due to nonuniform weighting for all but the smallest of systems.

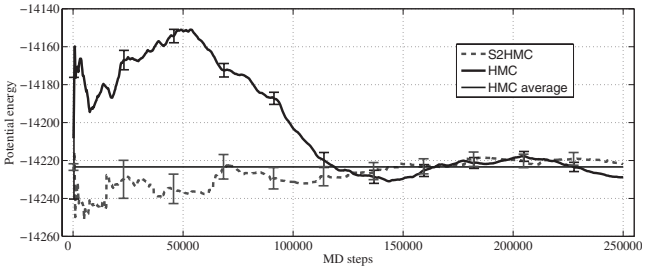

FIG. 2. Average potential energy for 4002 atom water model. The error bars represent 1 SD, estimated as discussed in Sec. III C. The average HMC value was obtained from a total of 156 configurations generated from six different initial conditions and 25 different seeds for generating initial momenta from each initial condition ($6+6\times25=156$ configurations). Each initial condition was taken every 100 000 steps from an original simulation, and then each one of the remaining 150 simulations was run for additional 100 000 steps.

We carried out further tests to estimate the average potential energy for the 4002 water box model. By taking six configurations produced from the above tests at 100 000 step intervals and running 25 simulations of 100 000 steps for each one using different seeds for generating momenta, an average was determined as $-14\ 223.4$ kcal mol^{-1}. There is thus a total of $6+6\times25=156$ configurations used to compute the average. This result is shown in Fig. 2 for comparison to the above tests.

VI. CONCLUSIONS

The S2HMC method based on a processed leapfrog method that has a fourth order *separable* shadow Hamiltonian overcomes the limitations of SHMC by removing the need to generate momenta from a nonseparable PDF. S2HMC can thus achieve the theoretical maximum speedup over HMC methods using fourth-order integrators, at the cost of a second-order leapfrog. This was proven analytically and experimentally with water boxes from 1002 to 16002 atoms. The efficiency of S2HMC in common with HMC is largely independent of the MD trajectory. S2HMC gives over a factor of 2 speedup over HMC in our tests.

S2HMC is thus a rigorous method that needs no fine tuning of parameters, a significant advantage over the original SHMC. The ideas behind S2HMC could be combined with generalizations of SHMC, such as TSHMC and GSHMC, which would allow other ensembles and simulation types to be performed, such as those with weak thermostating. Significantly, since the biased distribution of S2HMC and the target distribution are very close, there is no statistical inefficiency. The main contributor to the speedup of S2HMC in estimating statistics comes from the enhanced sampling of configuration space.

We envision that S2HMC and potential generalizations would be most useful in applications that require high accuracy. The implementation of S2HMC and an optimized SHMC used here are available in the program PROTOMOL 2.1. The Python implementation used to present the method can be found in http://mdlab.sourceforge.net/s2hmc.

ACKNOWLEDGMENTS

This material is based upon the work supported by the National Science Foundation (Grant Nos. DBI-0450067 and CCF-0135195) and the National Institute of General Medical Sciences (Grant No. R01GM083605).

[1] S. Duane, A. D. Kennedy, B. J. Pendleton, and D. Roweth, Phys. Lett. B **195**, 216 (1987).
[2] A. Brass, B. J. Pendleton, Y. Chen, and B. Robson, Biopolymers **33**, 1307 (1993).
[3] B. Mehlig, D. W. Heermann, and B. M. Forrest, Phys. Rev. B **45**, 679 (1992).
[4] M. Creutz, Phys. Rev. D **38**, 1228 (1988).
[5] A. D. Kennedy and B. Pendleton, Nucl. Phys. B, Proc. Suppl. **20**, 118 (1991).
[6] M. López-Marcos, J. M. Sanz-Serna, and R. D. Skeel, *Numerical Analysis 1995* (Longmans, Green, New York, 1996), pp. 107–122.
[7] E. Hairer, C. Lubich, and G. Wanner, *Geometric Numerical Integration: Structure-Preserving Algorithms for Ordinary Differential Equations*, Springer Series in Computational Mathematics Vol. 31 (Springer-Verlag, Berlin, 2002).
[8] J. A. Izaguirre and S. S. Hampton, J. Comput. Phys. **200**, 581 (2004).
[9] E. Akhmatskaya and S. Reich, *New Algorithms for Macromolecular Simulation* (Springer-Verlag, Berlin, 2006), Vol. 49, pp. 141–151.
[10] C. L. Wee, M. S. Sansom, S. Reich, and E. Akhmatskaya, J. Phys. Chem. B **112**, 5710 (2008).
[11] E. Akhmatskaya and S. Reich, J. Comput. Phys. **227**, 4934 (2008).
[12] E. Akhmatskaya, N. Bou-Rabee, and S. Reich, J. Comput. Phys. **228**, 2256 (2009).
[13] T. Matthey, T. Cickovski, S. S. Hampton, A. Ko, Q. Ma, M. Nyerges, T. Raeder, T. Slabach, and J. A. Izaguirre, ACM Trans. Math. Softw. **30**, 237 (2004).
[14] L. Verlet, Phys. Rev. **159**, 98 (1967).
[15] T. Cickovski, C. Sweet, and J. A. Izaguirre, Proceedings of the 40th Annual Simulation Symposium, 2007, pp. 256–266.
[16] R. D. Skeel and D. J. Hardy, SIAM J. Sci. Comput. (USA) **23**, 1172 (2001).
[17] R. D. Engle, R. D. Skeel, and M. Drees, J. Comput. Phys. **206**, 432 (2005).
[18] S. Blanes, S. Casas, and A. Murua, SIAM (Soc. Ind. Appl. Math.) J. Numer. Anal. **42**, 531 (2004).
[19] T. Littell, R. Skeel, and M. Zhang, SIAM (Soc. Ind. Appl. Math.) J. Numer. Anal. **34**, 1792 (1997).
[20] P. R. Bevington and D. K. Robinson, *Data Reduction and Error Analysis for the Physical Sciences* (McGraw-Hill, New York, 2003).
[21] M. E. Nelson and J. M. Bower, Trends Neurosci. **13**, 403 (1990).
[22] A. D. MacKerell, Jr., D. Bashford, M. Bellott, R. L. Dunbrack, Jr., J. Evanseck, M. J. Field, S. Fischer, J. Gao, H. Guo, S. Ha, D. Joseph, L. Kuchnir, K. Kuczera, F. T. K. Lau, C. Mattos, S. Michnick, T. Ngo, D. T. Nguyen, B. Prodhom, B. Roux, M. Schlenkrich, J. Smith, R. Stote, J. Straub, M. Watanabe, J. Wiorkiewicz-Kuczera, D. Yin, and M. Karplus, FASEB J. **A143**, 6 (1992).
[23] A. D. MacKerell, Jr., D. Bashford, M. Bellott, R. L. Dunbrack, Jr., J. Evanseck, M. J. Field, S. Fischer, J. Gao, H. Guo, S. Ha, D. Joseph, L. Kuchnir, K. Kuczera, F. T. K. Lau, C. Mattos, S. Michnick, T. Ngo, D. T. Nguyen, B. Prodhom, I. W. E. Reiher, B. Roux, M. Schlenkrich, J. Smith, R. Stote, J. Straub, M. Watanabe, J. Wiorkiewicz-Kuczera, D. Yin, and M. Karplus, J. Phys. Chem. B **102**, 3586 (1998).
[24] W. L. Jorgensen, J. Chandrasekhar, J. D. Madura, R. W. Impey, and M. L. Klein, J. Chem. Phys. **79**, 926 (1983).