Fast update algorithm for the quantum Monte Carlo simulation of the Hubbard model

Phani K. V. V. Nukala,[1] Thomas A. Maier,[1] Michael S. Summers,[1] Gonzalo Alvarez,[1] and Thomas C. Schulthess[1,2]
[1]*Computer Science and Mathematics Division, Oak Ridge National Laboratory, Oak Ridge, Tennessee 37831-6164, USA*
[2]*Institut für Theoretische Physik, ETH Zürich, 8093 Zürich, Switzerland*

(Received 21 April 2009; revised manuscript received 8 October 2009; published 17 November 2009)

This paper presents an efficient algorithm for computing the transition probability in auxiliary field quantum Monte Carlo simulations of strongly correlated electron systems using a Hubbard model. This algorithm is based on a low rank updating of the underlying linear algebra problem, and results in significant computational savings. The computational complexity of computing the transition probability and Green's function update reduces to $\mathcal{O}(k^2)$ during the kth step, where k is the number of accepted spin flips, and results in an algorithm that is faster than the competing delayed update algorithm. Moreover, this algorithm is orders of magnitude faster than traditional algorithms that use naive updating of the Green's function matrix.

DOI: 10.1103/PhysRevB.80.195111 PACS number(s): 02.70.Ss, 71.27.+a, 71.10.Fd

I. INTRODUCTION

The description of solids at the atomistic level is complicated due to the interaction among many of its constituents, namely ions or electrons. The corresponding solution of the quantum many-body problem is cumbersome and becomes computationally intensive, if not intractable. Consequently, simplified models are in common use to gain valuable insight into the description of solids. An important model for describing strongly correlated electron systems is the Hubbard model,[1] which describes interacting electrons in narrow energy bands, and which has been applied to problems as diverse as high-T_c superconductivity, band magnetism, and the metal-insulator transition.

In the Hubbard model,[1] the Hamiltonian has the form

$$\mathcal{H} = -t\sum_{\langle i,j\rangle} c^{\dagger}_{i\sigma}c_{j\sigma} + U\sum_{i} n_{i\uparrow}n_{i\downarrow}, \tag{1}$$

where t is the hopping integral and U is the on-site Coulomb repulsion. The fermionic operators $c^{\dagger}_{i\sigma}$ and $(c_{i\sigma})$, respectively, create and destroy an electron on-site i with spin σ, and $n_{i\sigma}=c^{\dagger}_{i\sigma}c_{i\sigma}$ is the corresponding number operator. The first term describes the hopping of electrons between nearest neighbor sites i and j, as denoted by $\langle\cdots\rangle$, and the second term describes the on-site Coulomb repulsion between two electrons. In the above form, the Hubbard model describes the competition between (i) itinerancy as described by the hopping term, and (ii) localization as described by the on-site Coulomb repulsion term.

Traditionally, the Blankenbecler-Scalapino-Sugar (BSS) quantum Monte Carlo algorithm[2,3] has been used to solve the Hubbard model on a finite lattice. Important progress has been made recently in simulations of the Hubbard model using dynamical mean field (DMFT) and quantum cluster simulations.[4–6] The numerically expensive part of these simulations is the solution of an embedded cluster problem. Quantum Monte Carlo methods such as the Hirsch-Fye auxiliary field algorithm[7] and the recently developed continuous time diagrammatic methods[8,9] allow the solution of large clusters. In the following, we will use the Hirsch-Fye (HF) cluster quantum Monte Carlo (QMC) method[10] to illustrate the new algorithm, and note that its application to the BSS QMC algorithm[2] is straightforward, given the similarity of the equations for the transition probability and the Green's function updates[3] to those in the Hirsch-Fye algorithm.

The cluster HF-QMC algorithm employs a path-integral formalism on a space- (imaginary) time lattice of size $N=N_cN_l$, where N_c and N_l represent the cluster size and the number of time slices (time steps) used in the path integral, respectively.[10] The canonical Hirsch-Hubbard-Stratonovich (HHS) transformation[11] is used to replace the interaction term in the Hamiltonian with a bilinear term and an additional sum over auxiliary degrees of freedom (known as the HHS field) at every point in space time. The HS spin fields are integrated using a Monte Carlo method that sweeps through the space-time lattice by attempting to flip the HS spin at each of the lattice sites. This local change (flipping of the spin at site p, i.e., $s_p\mapsto -s_p$) from configuration ψ to ψ' is accepted depending on the ratio r' given by

$$r' = \frac{\rho(\psi')}{\rho(\psi)} = \frac{\det \mathbf{G}^{-1}_{\uparrow}(\psi')\det \mathbf{G}^{-1}_{\downarrow}(\psi')}{\det \mathbf{G}^{-1}_{\uparrow}(\psi)\det \mathbf{G}^{-1}_{\downarrow}(\psi)} = R_{\uparrow}R_{\downarrow}, \tag{2}$$

where ρ is a *density matrix* that describes the weight of a HS spin configuration, $\mathbf{G}$ is the Green's function matrix of size $N\times N$ and R_σ with $\sigma=\uparrow,\downarrow$ defines the ratio of fermion determinants as

$$R_\sigma = \frac{\det \mathbf{G}^{-1}_{\sigma}(\psi')}{\det \mathbf{G}^{-1}_{\sigma}(\psi)}. \tag{3}$$

Using this model, the simulation proceeds by visiting each location of the lattice and proposing a local change (event). The probability that the change is accepted is based on the ratio r' in Eq. (2), which requires the computation of the determinant of the Green's function matrix $\mathbf{G}^{-1}_{\sigma}(\psi')$ in the ψ' configuration. The configuration ψ' is accepted with probability $r=r'/(1+r')$ (*heat-bath algorithm*). That is, the proposed local change is accepted and the system configuration changes from $\psi\mapsto\psi'$ when an arbitrarily chosen random number is less than r. If not, the local change is rejected and the system remains in configuration ψ. Each time a local change is accepted, the Green's function matrix $\mathbf{G}(\psi)$ is modified to $\mathbf{G}(\psi')$ by a low-rank update and the simulation proceeds through proposing a local change, which requires the recomputation of the determinant of the modified Green's function matrix $\mathbf{G}(\psi'')$ in the subsequent configuration ψ''. This progression of simulation through local changes proceeds for many steps until the observables converge to the desired accuracy of the Monte Carlo procedure.[12]

Large scale simulations using the Hubbard model have often been hampered due to the fact that ratios of fermionic determinants R_σ need to be calculated for every proposed spin flip, and the Green's function matrix needs to be updated whenever the proposed spin flip is accepted. In typical quantum cluster calculations,[5] a significant part of the simulation time is spent in computing and updating the successive Green's function matrices. Traditionally, the Sherman-Morrison formula[13] is used to update the Green's function matrix [see Eq. (4), which requires $\mathcal{O}(N^2)$ computations, where N is the size of the Green's function matrix]. These traditional algorithms that employ repetitive computation techniques, wherein the linear algebra subproblem (Sherman-Morrison update) is repetitively computed during each of the simulation steps, pose a significant computational challenge even for modern supercomputers since a large number of MC steps are required to solve the problem. In a typical Monte Carlo based simulation, the number of successive Green's function updates or steps, N_{steps}, is of the order of millions and increases with increasing lattice system sizes. In addition to the limitation due to the infamous fermion sign problem, it is for this reason that simulations of large lattice systems are not possible despite the fact that such large scale simulations are necessary to develop a better understanding of relevant physics.

However, since each of the successive Green's function matrices differ by a low-rank update, an updating scheme of some kind is likely to be more efficient than employing a repetitive computational technique (Sherman-Morrison update) on each of these successive matrices. Recently, a delayed Green's function updating algorithm[14] was proposed that reduced the computational complexity to $\mathcal{O}(kN)$ during the k-th step [compare this with $\mathcal{O}(N^2)$ required using Sherman-Morrison update]. Using this delayed update algorithm, the updated Green's matrix is computed using the efficient double-precision general matrix-matrix multiply (DGEMM)[15] rather than the expensive double-precision general vector-vector outer product (DGER)[15]. This delayed updating algorithm resulted in significant computational savings compared to traditional algorithms.

This paper presents a more efficient algorithm that is faster than the competing delayed update algorithm used in the simulation of the Hubbard model. Specifically, the proposed submatrix update algorithm reduces the computational complexity of computing the transition probability and Green's function update to $\mathcal{O}(k^2)$ during the k-th step compared to $\mathcal{O}(kN)$ complexity of delayed update scheme (the prefactor is smaller). When compared to traditional algorithms based on Sherman-Morrison updates, the proposed submatrix update algorithm is orders of magnitude faster than those that use naive updating of the Green's function matrix. Note, however, that the systematic error due to time discretization inherent to the HF and BSS QMC algorithms is not removed by the proposed algorithm.

The organization of the paper is as follows. In Sec. II, we present the computational problem and the traditional algorithms that are used in the computation. Sec. III presents the submatrix update algorithm and the comparison of these various approaches are discussed in Sec. IV. Conclusions are presented in Sec. V. In the following, we use the notation of capital bold face letters to denote matrices and small bold face letters to denote vectors.

II. TRADITIONAL ALGORITHMS

Whenever a local change in spins $(s_p \mapsto s_p')$ at site p is proposed, the Green's function matrix in the configuration [$(k+1)$th step] is given by

$$\mathbf{G}_{k+1} = \mathbf{G}_k + \alpha_k(\mathbf{G}_k(:,p) - \mathbf{e}_p) \otimes \mathbf{G}_k(p,:), \tag{4}$$

where $\mathbf{G}_k$ and $\mathbf{G}_{k+1}$ are the Green's function matrices at the kth and $(k+1)$th steps, $\mathbf{G}_k(:,p)$ and $\mathbf{G}_k(p,:)$ denote the pth column and pth row of $\mathbf{G}_k$ respectively, $\mathbf{e}_p = (0,\ldots,1,\ldots,0)^t$ denotes a unit vector with 1 on the pth entry and 0 everywhere else, $\otimes$ denotes the vector outer product such that $\mathbf{a}\otimes\mathbf{b}=\mathbf{a}\mathbf{b}^t$ for any set of vectors $\mathbf{a}$ and $\mathbf{b}$, and

$$\alpha_k = \frac{\gamma_k}{1+[1-\mathbf{G}_k(p,p)]\gamma_k}. \tag{5}$$

In the above equation, $\mathbf{G}_k(p,p)$ is the p-th diagonal entry of $\mathbf{G}_k$ and $\gamma_k = (\exp[-\lambda\sigma(s_p - s_p')] - 1)$. The determinant of $\mathbf{G}_{k+1}$ is given by

$$\begin{aligned}\det(\mathbf{G}_{k+1}) &= \det(\mathbf{G}_k)\{1+\alpha_k[\mathbf{G}_k(p,p)-1]\}\\ &= \det(\mathbf{G}_k)\{1+\gamma_k[1-\mathbf{G}_k(p,p)]\}^{-1},\end{aligned} \tag{6}$$

and hence the ratio of determinants R_k is given by

$$R_k = \frac{\det(\mathbf{G}_{k+1})}{\det(\mathbf{G}_k)} = \frac{1}{\{1+\gamma_k[1-\mathbf{G}_k(p,p)]\}}. \tag{7}$$

Hence, for any given k, a trivial, straight-forward computation of R_k can be obtained in $\mathcal{O}(N^2)$ computations by first updating $\mathbf{G}_k \mapsto \mathbf{G}_{k+1}$ as in Eq. (4), which requires $\mathcal{O}(N^2)$ computations for each Green's function matrix update, and then computing R_k using Eq. (7). However, repetitive computation of R_k during each of the MC simulation steps (for $k=0,1,2,\ldots$) requires an efficient procedure to compute successive R_k [or alternatively $\mathbf{G}_k(p,p)$]. Since we only require the determinant ratio R_k to accept or reject a Monte Carlo step, a delayed Green's function update algorithm can be adopted that would reduce the computational complexity of evaluating R_k during the k-th step to $\mathcal{O}(kN)$ instead of $\mathcal{O}(N^2)$ required during the straight-forward updating of $\mathbf{G}$ using Eq. (4). The delayed updating algorithm for the $(k+1)$th step is given by Algorithm 1. The additional storage requirements during each step are two vectors of size N. In the Algorithm 1, symbol $\odot$ denotes term-wise multiplication of vectors such that $\mathbf{c}=\mathbf{a}\odot\mathbf{b}$ implies $c_i = a_i b_i$, where a_i, b_i, and c_i denote the ith components of vectors $\mathbf{a}$, $\mathbf{b}$, and $\mathbf{c}$ respectively. Also, $\mathbf{G}_0$ in Algorithm 1 refers to the Green's function at step 0 and is not related to the bare Green's function matrix.

Algorithm 1 Delayed Green's Function Updating Algorithm

```
1: Given $\mathbf{G}_0$ and $\mathbf{d}_0=\mathrm{diag}(\mathbf{G}_0)$
2: Compute $\gamma_k=(\exp^{-\lambda\sigma(s_p-s'_p)}-1)$
3: Compute $R=\frac{1}{1+(1-\mathbf{d}_k(p))\gamma_k}$
4: Compute $\alpha_k=R\gamma_k$
5: Set $\mathbf{a}_k=\mathbf{G}_0(:,p)$, and $\mathbf{b}_k=\mathbf{G}_0(p,:)$
6: for $i=0$ to $k-1$ do
7:     Compute $\mathbf{a}_k=\mathbf{a}_k+\mathbf{b}_i(p)\mathbf{a}_i$
8:     Compute $\mathbf{b}_k=\mathbf{b}_k+\mathbf{a}_i(p)\mathbf{b}_i$
9: end for
10: Update $\mathbf{a}_k=\alpha_k(\mathbf{a}_k-\mathbf{e}_p)$
11: Update $\mathbf{d}_k=\mathbf{d}_k+\mathbf{a}_k\odot\mathbf{b}_k$
```

Since the computational cost of kth step increases as $\mathcal{O}(kN)$, Algorithm 1 requires the occasional Green's function matrix updates. This is especially the case whenever the cost of computing ℓ additional steps (beyond the current k steps) using Algorithm 1 becomes comparable to or exceeds the cost of a matrix-matrix product involved in the Green's function matrix update. At this stage, updating of Green's function matrix is done by

$$\mathbf{G}_q=\mathbf{G}_0+\mathbf{X}_{q-1}\mathbf{Y}^t_{q-1}, \tag{8}$$

where $\mathbf{X}_{q-1}=[\mathbf{a}_0|\mathbf{a}_1|\cdots|\mathbf{a}_{q-1}]$, $\mathbf{Y}_{q-1}=[\mathbf{b}_0|\mathbf{b}_1|\cdots|\mathbf{b}_{q-1}]$, and q is the number of rank 1 updates allowed between the full Green's function matrix updates.

An alternative to the delayed Green's function matrix update algorithm may be formulated by expressing Eq. (4) as

$$\mathbf{G}_{k+1}=[\mathbf{I}+\mathbf{a}_k\otimes\mathbf{e}_p]\mathbf{G}_k, \tag{9}$$

where $\mathbf{a}_k=\alpha_k[\mathbf{G}_k(:,p)-\mathbf{e}_p]$. Consequently, for each additional step, we require storage space for a vector $\mathbf{a}_k$ of size N. In addition, we need to store an index variable p_k that maps $k\mapsto p$. This can be conveniently stored by defining an index vector $\mathbf{p}$ such that $\mathbf{p}(k)=p$. Based on Eq. (9), a recursive scheme for computing $\mathbf{G}_{k+1}$ may be formulated as

$$\begin{aligned}\mathbf{G}_{k+1}&=(\mathbf{I}+\mathbf{a}_k\otimes\mathbf{e}_{\mathbf{p}(k)})\ldots(\mathbf{I}+\mathbf{a}_0\otimes\mathbf{e}_{\mathbf{p}(0)})\mathbf{G}_0\\&=\left[\prod_{j=0}^{k}(\mathbf{I}+\mathbf{a}_j\otimes\mathbf{e}_{\mathbf{p}(j)})\right]\mathbf{G}_0.\end{aligned} \tag{10}$$

An $\mathcal{O}(kN)$ recursive algorithm based on Eq. (10) may be formulated; however, similar to the delayed updating algorithm, this algorithm will also require occasional Green's function matrix updates as k approaches N.

III. SUBMATRIX UPDATE ALGORITHM

In the following, we introduce an algorithm that reduces the computational complexity during k-th step to $\mathcal{O}(k^2)$ instead of $\mathcal{O}(kN)$. For $k\ll N$, this algorithm is expected to result in significant computational savings. For demonstration purposes, it is convenient to describe the algorithm using the inverse of the Green's function matrices although such inverses are never actually computed. The main advantage of such an approach is that the rank 1 updates of Green's function matrices reduce to a single column update of the inverse of Green's function matrix. Let $\mathbf{A}_k=\mathbf{G}_k^{-1}$ and $\mathbf{A}_{k+1}=\mathbf{G}_{k+1}^{-1}$. Then, using Sherman-Morrison formula, Eq. (4) can be rewritten as

$$\begin{aligned}\mathbf{A}_{k+1}&=\mathbf{A}_k+\gamma_k(\mathbf{A}_k(:,p)-\mathbf{e}_p)\otimes\mathbf{e}_p\\&=\mathbf{A}_k+\gamma_k\mathbf{A}_k(:,p)\otimes\mathbf{e}_p-\gamma_k\mathbf{e}_p\otimes\mathbf{e}_p.\end{aligned} \tag{11}$$

The updating of $\mathbf{A}_k$ in Eq. (11) can be interpreted as a multiplication of pth column of $\mathbf{A}_k$ by $(1+\gamma_k)$ followed by a subtraction of γ_k from the pth diagonal element. Symbolically we represent Eq. (11) as

$$\mathbf{A}_{k+1}=\mathbf{A}_k+\begin{bmatrix}\gamma_k\\\times\\\times\\\times\\\times\\\times\end{bmatrix}-\begin{bmatrix}&p&&&\\&|&&&\\-&\gamma_k&-&-&-\\&|&&&\\&|&&&\\&|&&&\end{bmatrix}, \tag{12}$$

where the second term is understood to multiply the pth column of $\mathbf{A}_k$ by γ_k and the third term represents a nonzero pth diagonal entry of γ_k.

Denoting $\widetilde{\mathbf{A}}_k=\mathbf{A}_k+\gamma_k\mathbf{A}_k(:,p)\otimes\mathbf{e}_p$, which is equivalent to updating the pth column of $\mathbf{A}_k$ by multiplying it by $(1+\gamma_k)$, Eq. (11) can be rewritten as

$$\mathbf{A}_{k+1}=\widetilde{\mathbf{A}}_k-\gamma_k\mathbf{e}_p\otimes\mathbf{e}_p. \tag{13}$$

It should be noted that the inverse $\widetilde{\mathbf{G}}_k=\widetilde{\mathbf{A}}_k^{-1}$ is equivalent to updating the pth row of $\mathbf{G}_k$ by dividing it by $(1+\gamma_k)$. Consequently,

$$\det(\widetilde{\mathbf{A}}_k)=(1+\gamma_k)\det(\mathbf{A}_k), \tag{14}$$

$$\det(\widetilde{\mathbf{G}}_k)=\frac{1}{(1+\gamma_k)}\det(\mathbf{G}_k). \tag{15}$$

Based on Eq. (13), the $\det(\mathbf{A}_{k+1})$ can be computed as

$$\begin{aligned}\det(\mathbf{A}_{k+1})&=\det(\widetilde{\mathbf{A}}_k)\det(\mathbf{I}-\gamma_k\mathbf{e}_p^t\widetilde{\mathbf{A}}_k^{-1}\mathbf{e}_p)\\&=\det(\mathbf{A}_k)(1+\gamma_k)(1-\gamma_k\mathbf{e}_p^t\widetilde{\mathbf{G}}_k\mathbf{e}_p)\\&=\det(\mathbf{A}_k)(1+\gamma_k)\left(1-\frac{\gamma_k}{1+\gamma_k}\mathbf{G}_k(p,p)\right)\\&=-\det(\mathbf{A}_k)\gamma_k\left(\mathbf{G}_k(p,p)-\frac{1+\gamma_k}{\gamma_k}\right).\end{aligned} \tag{16}$$

Based on the above description, a recursive update of Green's function matrix $\mathbf{G}_{k+1}$ over $(k+1)$ steps may be formulated as an update of $\mathbf{A}_{k+1}$ as

$$\begin{aligned}\mathbf{A}_{k+1}&=\mathbf{A}_0+\sum_{j=0}^{k}\gamma_j\{\mathbf{A}_0[:,\mathbf{p}(j)]-\mathbf{e}_{\mathbf{p}(j)}\}\otimes\mathbf{e}_{\mathbf{p}(j)}\\&=\widetilde{\mathbf{A}}_k-\sum_{j=0}^{k}\gamma_j\mathbf{e}_{\mathbf{p}(j)}\otimes\mathbf{e}_{\mathbf{p}(j)}=\widetilde{\mathbf{A}}_k-\mathbf{X}_k\mathbf{Y}_k^t,\end{aligned} \tag{17}$$

where $\mathbf{p}$ is the index vector as defined before, $\mathbf{X}_k = [\gamma_0 \mathbf{e}_{\mathbf{p}(0)}|\cdots|\gamma_k \mathbf{e}_{\mathbf{p}(k)}]$ and $\mathbf{Y}_k=[\mathbf{e}_{\mathbf{p}(0)}|\cdots|\mathbf{e}_{\mathbf{p}(k)}]$, which are never actually stored but are used here as a notational convenience. Symbolically, this translates to

$$\mathbf{A}_{k+1} = \mathbf{A}_0 + \begin{bmatrix} \gamma_0 & & \gamma_k \\ \times & & \times \\ \times & & \times \\ \times & \cdots & \times \\ \times & & \times \\ \times & & \times \end{bmatrix} - \begin{bmatrix} & \mathbf{p}(0) & & \mathbf{p}(k) & \\ & | & & | & \\ - & \gamma_0 & - & - & - \\ & | & & | & \\ - & - & - & \gamma_k & - \\ & | & & | & \end{bmatrix}. \tag{18}$$

Following earlier description, we have $\tilde{\mathbf{G}}_k=\tilde{\mathbf{A}}_k^{-1}$, which is obtained by dividing each of the $\mathbf{p}(j)$ rows of $\mathbf{G}_0$ by $(1+\gamma_j)$, and

$$\det(\tilde{\mathbf{A}}_k) = \left[\prod_{j=0}^{k} (1+\gamma_j)\right] \det(\mathbf{A}_0), \tag{19}$$

$$\det(\tilde{\mathbf{G}}_k) = \left[\prod_{j=0}^{k} \frac{1}{(1+\gamma_j)}\right] \det(\mathbf{G}_0). \tag{20}$$

Based on Eq. (17) and noting that $\tilde{\mathbf{G}}_k=\tilde{\mathbf{A}}_k^{-1}$, the $\det(\mathbf{A}_{k+1})$ can be expressed as

$$\det(\mathbf{A}_{k+1}) = \det(\tilde{\mathbf{A}}_k)\det(\mathbf{I}-\mathbf{Y}_k^t\tilde{\mathbf{G}}_k\mathbf{X}_k). \tag{21}$$

For notational convenience, let us define $\tilde{\mathbf{G}}_k(\mathbf{p})$ as

$$\tilde{\mathbf{G}}_k(\mathbf{p}) = \begin{Bmatrix} \tilde{\mathbf{G}}_0[\mathbf{p}(0),\mathbf{p}(0)] & \cdots & \tilde{\mathbf{G}}_0[\mathbf{p}(0),\mathbf{p}(k)] \\ \vdots & \ddots & \vdots \\ \tilde{\mathbf{G}}_0[\mathbf{p}(k),\mathbf{p}(0)] & \cdots & \tilde{\mathbf{G}}_0[\mathbf{p}(k),\mathbf{p}(k)] \end{Bmatrix}. \tag{22}$$

The matrix $\mathbf{G}_k(\mathbf{p})$ is defined similarly. Using this notation, we have

$$\det(\mathbf{I}-\mathbf{Y}_k^t\tilde{\mathbf{G}}_k\mathbf{X}_k) = \det\left[\mathbf{I} - \tilde{\mathbf{G}}_k(\mathbf{p})\begin{bmatrix} \gamma_0 & & \\ & \ddots & \\ & & \gamma_k \end{bmatrix}\right]$$
$$= \det\left[\mathbf{I} - \begin{bmatrix} \frac{1}{1+\gamma_0} & & \\ & \ddots & \\ & & \frac{1}{1+\gamma_k} \end{bmatrix} \mathbf{G}_k(\mathbf{p}) \times \begin{bmatrix} \gamma_0 & & \\ & \ddots & \\ & & \gamma_k \end{bmatrix}\right], \tag{23}$$

which is further simplified as

$$\det(\mathbf{I}-\mathbf{Y}_k^t\tilde{\mathbf{G}}_k\mathbf{X}_k) = (-1)^{k+1}\left(\prod_{j=0}^{k}\frac{\gamma_j}{1+\gamma_j}\right)\det(\mathbf{\Gamma}_k), \tag{24}$$

where

$$\mathbf{\Gamma}_k = \begin{Bmatrix} \mathbf{G}_0[\mathbf{p}(0),\mathbf{p}(0)] - \frac{1+\gamma_0}{\gamma_0} & \cdots & \mathbf{G}_0[\mathbf{p}(0),\mathbf{p}(k-1)] & \mathbf{G}_0[\mathbf{p}(0),\mathbf{p}(k)] \\ \vdots & \ddots & \vdots & \vdots \\ \mathbf{G}_0[\mathbf{p}(k-1),\mathbf{p}(0)] & \cdots & \mathbf{G}_0[\mathbf{p}(k-1),\mathbf{p}(k-1)] - \frac{1+\gamma_{k-1}}{\gamma_{k-1}} & \mathbf{G}_0[\mathbf{p}(k-1),\mathbf{p}(k)] \\ \mathbf{G}_0[\mathbf{p}(k),\mathbf{p}(0)] & \cdots & \mathbf{G}_0[\mathbf{p}(k),\mathbf{p}(k-1)] & \mathbf{G}_0[\mathbf{p}(k),\mathbf{p}(k)] - \frac{1+\gamma_k}{\gamma_k} \end{Bmatrix}. \tag{25}$$

It should be noted that $\mathbf{\Gamma}_k$, which is obtained by subtracting a diagonal matrix with entries $\frac{1+\gamma_i}{\gamma_i}$ for $i=0,1,2,\ldots,k$ from $\mathbf{G}_k(\mathbf{p})$, is never really computed or stored; instead, it is stored in the form of a *LU* decomposition, which readily enables the computation of $\det(\mathbf{\Gamma}_k)$ in Eq. (24). Once we have an *LU* decomposition of $\mathbf{G}_k(\mathbf{p})$, the determinant of $\mathbf{G}_k(\mathbf{p})$ is equal to the product of diagonal entries of $\mathbf{U}$ factor. Moreover, as shown below, it is relatively straightforward to update the *LU* factors of $\mathbf{G}_k(\mathbf{p})$ as k increases.

Symbolically, let us represent $\mathbf{\Gamma}_k$ as

$$\mathbf{\Gamma}_k = \begin{bmatrix} \mathbf{\Gamma}_{k-1} & \mathbf{s} \\ \mathbf{w}^t & d \end{bmatrix} = \begin{bmatrix} \mathbf{L}_{k-1} & \mathbf{0} \\ \mathbf{x}^t & 1 \end{bmatrix}\begin{bmatrix} \mathbf{U}_{k-1} & \mathbf{y} \\ \mathbf{0} & \beta \end{bmatrix}, \tag{26}$$

where $\mathbf{s}=\mathbf{G}_0[\mathbf{p}(0):\mathbf{p}(k-1),\mathbf{p}(k)]$, $\mathbf{w}^t=\mathbf{G}_0[\mathbf{p}(k),\mathbf{p}(0):\mathbf{p}(k-1)]$, and $d=\mathbf{G}_0[\mathbf{p}(k),\mathbf{p}(k)]-\frac{1+\gamma_k}{\gamma_k}$. Assuming that we have a *LU* decomposition of $\mathbf{\Gamma}_{k-1}=\mathbf{L}_{k-1}\mathbf{U}_{k-1}$, the *LU* factorization of $\mathbf{\Gamma}_k=\mathbf{L}_k\mathbf{U}_k$ can be obtained in $\mathcal{O}(k^2)$ operations by solving $\mathbf{L}_{k-1}\mathbf{y}=\mathbf{s}$, $\mathbf{U}_{k-1}^t\mathbf{x}=\mathbf{w}$, and $d=\beta+\mathbf{x}^t\mathbf{y}$. The $\det(\mathbf{\Gamma}_k)$ is then given by

$$\det(\mathbf{\Gamma}_k) = \beta\,\det(\mathbf{\Gamma}_{k-1}). \tag{27}$$

Combining Eqs. (19), (21), and (24), $\det(\mathbf{A}_{k+1})$ can be expressed as

$$\det(\mathbf{A}_{k+1}) = (-1)^{k+1}\left(\prod_{j=0}^{k}\gamma_j\right)\det(\mathbf{A}_0)\det(\mathbf{\Gamma}_k) = -\beta\gamma_k \det(\mathbf{A}_k). \tag{28}$$

Hence, $R_k = -\frac{1}{\beta\gamma_k}$, which is the desired result for computing the acceptance rate.

The submatrix update algorithm for the k-th step is presented in Algorithm 2, where we have used the short-hand notation $\mathbf{L}_{k-1}$ for $\mathbf{L}(1:k-1,1:k-1)$ and $\mathbf{U}_{k-1}$ for $\mathbf{U}(1:k-1,1:k-1)$. During the kth step, $\mathbf{L}_k$ and $\mathbf{U}_k$ are updated as given by Eq. (26).

Algorithm 2 Sub-Matrix Update Algorithm during k-th step ($s_p \mapsto s'_p$)

```
1: Given $\mathbf{G}_0$, and index vector $\mathbf{p}$ up to $k-1$ steps
2: Set $\mathbf{p}(k)=p$
3: Compute $\gamma_k=(\exp^{-\lambda\sigma(s_p-s'_p)}-1)$
4: Set $\mathbf{s}=\mathbf{G}_0[\mathbf{p}(0):\mathbf{p}(k-1),p]$
5: Set $\mathbf{w}^t=\mathbf{G}_0[p,\mathbf{p}(0):\mathbf{p}(k-1)]$
6: Set $d=\mathbf{G}_0(p,p)-\frac{1+\gamma_k}{\gamma_k}$
7: Solve $\mathbf{L}_{k-1}\mathbf{y}=\mathbf{s}$
8: Solve $\mathbf{U}^t_{k-1}\mathbf{x}=\mathbf{w}$
9: Compute $\beta=d-\mathbf{x}^t\mathbf{y}$
10: Compute $R_k=-\frac{1}{\beta\gamma_k}$
11: If the move is accepted, set $\mathbf{L}(k,1:k-1)=\mathbf{x}^t$ and
      $\mathbf{U}(1:k-1,k)=\mathbf{y}$
```

It should be noted that the **L** and **U** factors in Algorithm 2 are only updated if the move is accepted, in which case $k \mapsto k+1$. If not, k is not incremented and another Monte Carlo step is considered. We also note that the computation of R_k (alternatively, the acceptance rate) using Algorithm 2 is $\mathcal{O}(k^2)$ and is not constant in time as in traditional Hirsch-Fye algorithm.

In addition, similar to the delayed updating algorithm, the submatrix update algorithm also requires occasional Green's function matrix updates since the computational cost of the submatrix update algorithm increases as k approaches N. Using Eq. (17), the Green's function matrix $\mathbf{G}_q = \mathbf{A}_q^{-1}$ after q number of updates is given by

$$\begin{aligned}\mathbf{G}_q &= \mathbf{A}_q^{-1} = [\widetilde{\mathbf{A}}_{q-1} - \mathbf{X}_{q-1}\mathbf{Y}^t_{q-1}]^{-1} \\ &= \widetilde{\mathbf{A}}^{-1}_{q-1} + \widetilde{\mathbf{A}}^{-1}_{q-1}\mathbf{X}_{q-1}[\mathbf{I} - \mathbf{Y}^t_{q-1}\widetilde{\mathbf{A}}^{-1}_{q-1}\mathbf{X}_{q-1}]^{-1}\mathbf{Y}^t_{q-1}\widetilde{\mathbf{A}}^{-1}_{q-1},\end{aligned} \tag{29}$$

where we have used the Sherman-Morrison-Woodbury formula[13] in the last expression. After much simplifying, Eq. (29) can be shown to be

$$\mathbf{G}_q = \mathbf{D}^{-1}_{1+\gamma}[\mathbf{G}_0 - \mathbf{G}_0(:,\mathbf{p})\mathbf{\Gamma}^{-1}_{q-1}\mathbf{G}_0(\mathbf{p},:)], \tag{30}$$

where $\mathbf{\Gamma}^{-1}_{q-1} = \mathbf{U}^{-1}_{q-1}\mathbf{L}^{-1}_{q-1}$ is never really computed but used as a forward substitution (a lower triangular solve, see Ref. 13 for details) with $\mathbf{L}_{q-1}$ and a backward elimination (an upper triangular solve, see Ref. 13 for details) with $\mathbf{U}_{q-1}$, $\mathbf{G}_0(:,\mathbf{p})$, and $\mathbf{G}_0(\mathbf{p},:)$ denote respectively the columns and rows of $\mathbf{G}_0$ referred by the index array $\mathbf{p}$, and $\mathbf{D}^{-1}_{1+\gamma}$ is a unit diagonal matrix except for the entries of

$$\mathbf{D}^{-1}_{1+\gamma}[\mathbf{p}(k),\mathbf{p}(k)] = \frac{1}{1+\gamma_k}. \tag{31}$$

The computational cost of each of the submatrix updates is $\mathcal{O}(k^2)$, which implies that the total computational cost of k steps is $\mathcal{O}(k^3)$. However, the cost of resetting the Green's function matrix after q number of updates as in Eq. (30) is dominated by matrix-matrix multiplication whose computational cost is $\mathcal{O}(qN^2)$.

TABLE I. Computational cost (seconds) of various algorithms.

Size	Full	Delayed	Proposed
1000	7.023 ± 0.005	0.277 ± 0.004	0.015 ± 0.001
3000	194.76 ± 0.23	9.86 ± 0.01	0.49 ± 0.005

IV. NUMERICAL RESULTS

In order to compare the computational efficiencies of traditional (Sherman-Morrison), delayed (Algorithm 1), and submatrix update (Algorithm 2) algorithms, we test these algorithms on a randomly generated matrix $\mathbf{G}_0$. That is, since the algorithms are applicable for general matrices, we start with a matrix $\mathbf{G}_0$ whose elements are randomly chosen between 0 and 1. Then we consider rank 1 updates of $\mathbf{G}_0$ as given by Eq. (4) for m number of steps. The site locations p are randomly chosen for these m steps. Table I presents the computational timings obtained using the traditional full matrix updates via Sherman-Morrison formula, delayed updates and proposed updates via Algorithm 2. These results are presented for matrix sizes $N=1000$ and $N=3000$ with updates carried for $m=100$ and $m=300$ steps, respectively. The results are averaged over 100 simulations to achieve sufficient accuracy in the timing of these algorithms. These results indicate that the submatrix update algorithm is significantly faster than the delayed update algorithm, which in turn is significantly faster than the naive updating of the Green's matrices using the Sherman-Morrison formula. We note that exactly the same determinant ratios (R_k for $k=0,1,2,\ldots$) are obtained using each of the algorithms. In fact, the maximum difference between R_k for any $k=0,1,2,\ldots$ is of the order of 10^{-16}.

Although the above example does not consider multiple lattice sweeps (in fact $m<N$), it demonstrates the computational advantage that one could expect using the submatrix update algorithm whenever the average acceptance ratio in a MC simulation is small. In this sense, the above example only compares the computational cost of spin flip operations. The cost of the Green's function updates is not considered in this example. A more realistic Hubbard model simulation that includes the cost of the Green's function updates is considered below. In the following, we consider a typical 16-site

dynamic cluster quantum Monte Carlo simulation[5] of a two-dimensional Hubbard model with hopping t and Coulomb repulsion $U=4t$. The inverse temperature $\beta=\frac{40}{t}$. Here we used 300 time slices. In this case the initial Green's function corresponds to the noninteracting cluster embedded in a mean-field host and is given by a dense matrix of size $N=4800$. Since we are interested in the explicit comparison of different algorithms, we ran our tests using both delayed and submatrix algorithms with exactly the same initial conditions. These simulations account for both spin flips acceptances and rejections. Our simulations give identical results for both delayed and submatrix algorithms; however, the time to completion using submatrix algorithm is shorter than that using the delayed algorithm.[16] For comparison purposes, we did not consider the simulation using the traditional algorithm (Sherman-Morrison formula) since it requires a significant CPU time for each of the MC steps. Hence, the submatrix update algorithm is compared with the delayed update algorithm as presented in Algorithm 1.

Since the computational cost of kth MC step increases as k approaches N, we chose to update the Green's function matrix explicitly after a chosen q number of updates. That is, in the case of delayed update algorithm, we update $\mathbf{G}_0$ to $\mathbf{G}_q=\mathbf{G}_0+\mathbf{X}_{q-1}\mathbf{Y}^t_{q-1}$ using Eq. (8) after q number of updates. After each q number of updates, the process is reset with $\mathbf{G}_q\mapsto\mathbf{G}_0$ and $k=0$. This resetting is done by matrix-matrix multiplication and requires $\mathcal{O}(qN^2)$ operations. In the case of submatrix update algorithm, the updating is done by Eq. (30), which again involves a matrix-matrix multiplication with a leading order of $\mathcal{O}(qN^2)$ computations.

Figure 1 presents the CPU times taken by delayed and submatrix update algorithms for one complete lattice sweep. The timing results indicate that the submatrix algorithm is faster than the delayed update algorithm and that its computational cost does not increase with the reset size q. Moreover, Fig. 1(b) presents a detailed representation of CPU times taken by delayed and submatrix algorithms. The data indicate that the time taken for spin flip operations increases linearly for delayed algorithm while it remains constant and is extremely cheap for submatrix algorithm. On the contrary, the computational cost of the Green's function update is comparable for both algorithms, albeit delayed algorithm is faster due to less number of operations. A breakdown of operational cost of updating the Green's function matrix using the submatrix algorithm (Eq. (30)) is presented in Fig. 2. In this stacked chart, the additional computational cost of submatrix algorithm due to each of the left-hand side solve $[\mathbf{L}_k^{-1}\mathbf{G}_0(\mathbf{p},:)]$, right-hand side solve $[\mathbf{G}_0(:,\mathbf{p})\mathbf{U}_k^{-1}]$, and diagonal scaling with $\mathbf{D}^{-1}_{1+\gamma}$ are clearly shown. However, as expected, the computational cost associated with DGEMM would be same as that associated with delayed algorithm. Despite this additional computational cost in updating the Green's function matrix, the submatrix algorithm is faster overall due to significant efficiency gained during the spin flip operations.

V. CONCLUSIONS

We present an efficient algorithm for computing the transition probability in QMC simulations of strongly correlated

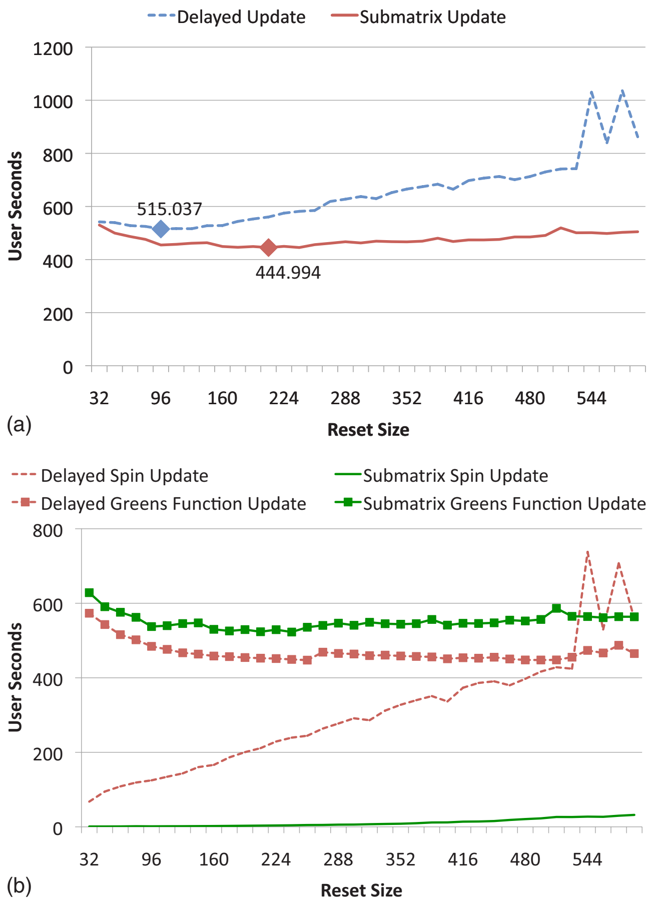

FIG. 1. (Color online) (a) CPU times taken by delayed and submatrix update algorithms for one complete lattice sweep for a variety of reset sizes q. System size is $N=4800$ and the total number of lattice site spin flips is $m=4800$. The Green's function matrix is updated every q steps. It is clear that the computational cost of submatrix algorithm is almost constant with the reset size q, whereas the cost of delayed algorithm increases linearly with reset size q. (b) A detailed representation of CPU times taken by delayed and submatrix algorithms. Figure shows the total CPU time taken for spin flip operations and the Green's function updates for a given reset size q. The time taken for spin-flip operations increases linearly for delayed algorithm while it remains constant and is extremely cheap for submatrix algorithm. On the contrary, the computational cost of the Green's function update is comparable for both algorithms, albeit delayed algorithm is faster due to less number of operations.

systems using the Hubbard model. The algorithm takes advantage of low-rank updates to the Green's function matrix in developing an efficient method to compute the transition probabilities. The present algorithm reduces the computational complexity of the kth MC step to $\mathcal{O}(k^2)$ compared to either $\mathcal{O}(kN)$ complexity using the competing delayed updating algorithm, or $\mathcal{O}(N^2)$ complexity using the traditional (Sherman-Morrison formula) updating schemes. The present algorithm is faster than the delayed updating scheme in terms of time to completion, and is orders of magnitude faster than the traditional schemes. Moreover, the computational cost of the present algorithm remains constant over a broad range of number of updates (q) between the Green's function resets,

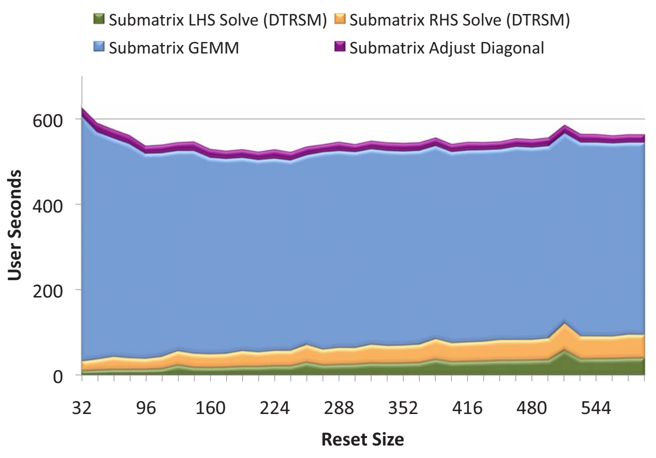

FIG. 2. (Color online) Further breakdown of CPU times taken by the submatrix algorithm in updating the Green's function update [Eq. (30)]. In the legend, LHS solve and RHS solve refer to $\mathbf{L}_k^{-1}\mathbf{G}_0(\mathbf{p},:)$ and $\mathbf{G}_0(:,\mathbf{p})\mathbf{U}_k^{-1}$ operations, respectively, and diagonal scaling refers to multiplication by $\mathbf{D}_{1+\gamma}^{-1}$. These are the additional costs associated with submatrix algorithm. The computational cost associated with DGEMM would be same as that associated with delayed algorithm.

which is in contrast with the delayed update algorithm. Furthermore, the algorithm can benefit from using effective bounds for the transition probability, which will further increase the efficiency of the algorithm. Estimation of these tighter bounds for transition probability will be considered in the future work.

ACKNOWLEDGMENTS

This research is sponsored by the Computational Science Research Division of the Office of Advanced Scientific Computing Research, the Scientific User Facilities Division of the Office of Basic Energy Sciences, and the Laboratory Directed Research and Development program of Oak Ridge National Laboratory. It is a pleasure to acknowledge stimulating discussions with Ed. D'Azevedo and Mark Jarrell.

APPENDIX

The algorithms presented earlier are derived for the case in which the Green's function matrix [Eq. (4)] is derived using a re-exponentiation formulation. Such a step involves an error of the order of $(\Delta\tau)^2$. Alternatively, using the original form of $\mathbf{G}$, the updating can be written as

$$\mathbf{G}_{k+1} = \mathbf{G}_k + \alpha_k \mathbf{G}_k(:,p) \otimes \mathbf{G}_k(\tilde{p},:), \tag{A1}$$

which can be rewritten as

$$\mathbf{A}_{k+1} = \mathbf{A}_k - \gamma_k \mathbf{e}_p \otimes \mathbf{e}_{\tilde{p}}. \tag{A2}$$

This can be interpreted as a subtraction of γ_k from the $(p,\tilde{p})$ element of the $\mathbf{A}_k$ matrix. Symbolically, Eq. (A2) can be represented as

$$\mathbf{A}_{k+1} = \mathbf{A}_k - \begin{bmatrix} & \tilde{p} & & & \\ & | & & & \\ - & \gamma_k & - & - & p \\ & | & & & \\ & | & & & \end{bmatrix}. \tag{A3}$$

Based on the above description, after $k+1$ number of updates, $\mathbf{A}_{k+1}$ can be expressed as

$$\mathbf{A}_{k+1} = \mathbf{A}_0 - \sum_{j=0}^{k} \gamma_j \mathbf{e}_{\mathbf{p}(j)} \otimes \mathbf{e}_{\tilde{\mathbf{p}}(j)} = \mathbf{A}_0 - \mathbf{X}_k \mathbf{Y}_k^t, \tag{A4}$$

where $\mathbf{p}$ is the index vector as defined before, $\mathbf{X}_k = [\gamma_0 \mathbf{e}_{\mathbf{p}(0)} | \ldots | \gamma_k \mathbf{e}_{\mathbf{p}(k)}]$ and $\mathbf{Y}_k = [\mathbf{e}_{\tilde{\mathbf{p}}(0)} | \ldots | \mathbf{e}_{\tilde{\mathbf{p}}(k)}]$. Symbolically, this translates to

$$\mathbf{A}_{k+1} = \mathbf{A}_0 - \begin{bmatrix} & \tilde{\mathbf{p}}(0) & & \tilde{\mathbf{p}}(k) & & \\ & | & & | & & \\ - & \gamma_0 & - & - & - & \mathbf{p}(0) \\ & | & & | & & \\ - & - & - & \gamma_k & - & \mathbf{p}(k) \\ & | & & | & & \end{bmatrix}. \tag{A5}$$

Based on Eq. (A5) and noting that $\mathbf{G}_0 = \mathbf{A}_0^{-1}$, the $\det(\mathbf{A}_{k+1})$ can be expressed as

$$\det(\mathbf{A}_{k+1}) = \det(\mathbf{A}_0)\det(\mathbf{I} - \mathbf{Y}_k^t \mathbf{G}_0 \mathbf{X}_k). \tag{A6}$$

For notational convenience, let us define $\mathbf{G}_k(\tilde{\mathbf{p}},\mathbf{p})$ as

$$\mathbf{G}_k(\tilde{\mathbf{p}},\mathbf{p}) = \begin{bmatrix} \mathbf{G}_0(\tilde{\mathbf{p}}(0),\mathbf{p}(0)) & \ldots & \mathbf{G}_0(\tilde{\mathbf{p}}(0),\mathbf{p}(k)) \\ \vdots & \ddots & \vdots \\ \mathbf{G}_0(\tilde{\mathbf{p}}(k),\mathbf{p}(0)) & \ldots & \mathbf{G}_0(\tilde{\mathbf{p}}(k),\mathbf{p}(k)) \end{bmatrix}. \tag{A7}$$

Using this notation, we have

$$\det(\mathbf{I} - \mathbf{Y}_k^t \mathbf{G}_0 \mathbf{X}_k) = \det\left[\mathbf{I} - \mathbf{G}_k(\tilde{\mathbf{p}},\mathbf{p}) \begin{bmatrix} \gamma_0 & & \\ & \ddots & \\ & & \gamma_k \end{bmatrix}\right], \tag{A8}$$

which is further simplified as

$$\det(\mathbf{I} - \mathbf{Y}_k^t \mathbf{G}_0 \mathbf{X}_k) = (-1)^{k+1} \left(\prod_{j=0}^{k} \gamma_j \right) \det(\boldsymbol{\Gamma}_k), \tag{A9}$$

where

$$\mathbf{\Gamma}_k = \begin{Bmatrix} \mathbf{G}_0[\tilde{\mathbf{p}}(0),\mathbf{p}(0)] - \frac{1}{\gamma_0} & \dots & \mathbf{G}_0[\tilde{\mathbf{p}}(0),\mathbf{p}(k-1)] & \mathbf{G}_0[\tilde{\mathbf{p}}(0),\mathbf{p}(k)] \\ \vdots & \ddots & \vdots & \vdots \\ \mathbf{G}_0[\tilde{\mathbf{p}}(k-1),\mathbf{p}(0)] & \dots & \mathbf{G}_0[\tilde{\mathbf{p}}(k-1),\mathbf{p}(k-1)] - \frac{1}{\gamma_{k-1}} & \mathbf{G}_0[\tilde{\mathbf{p}}(k-1),\mathbf{p}(k)] \\ \mathbf{G}_0[\tilde{\mathbf{p}}(k),\mathbf{p}(0)] & \dots & \mathbf{G}_0[\tilde{\mathbf{p}}(k),\mathbf{p}(k-1)] & \mathbf{G}_0[\tilde{\mathbf{p}}(k),\mathbf{p}(k)] - \frac{1}{\gamma_k} \end{Bmatrix}. \tag{A10}$$

Symbolically, let us represent $\mathbf{\Gamma}_k$ as

$$\mathbf{\Gamma}_k = \begin{bmatrix} \mathbf{\Gamma}_{k-1} & \mathbf{s} \\ \mathbf{w}^t & d \end{bmatrix} = \begin{bmatrix} \mathbf{L}_{k-1} & \mathbf{0} \\ \mathbf{x}^t & 1 \end{bmatrix} \begin{bmatrix} \mathbf{U}_{k-1} & \mathbf{y} \\ \mathbf{0} & \beta \end{bmatrix}, \tag{A11}$$

where $\mathbf{s}=\mathbf{G}_0(\tilde{\mathbf{p}}(0):\tilde{\mathbf{p}}(k-1),\mathbf{p}(k))$, $\mathbf{w}^t=\mathbf{G}_0(\tilde{\mathbf{p}}(k),\mathbf{p}(0):\mathbf{p}(k-1))$, and $d=\mathbf{G}_0(\tilde{\mathbf{p}}(k),\mathbf{p}(k))-\frac{1}{\gamma_k}$. Assuming that we have a *LU* decomposition of $\mathbf{\Gamma}_{k-1}=\mathbf{L}_{k-1}\mathbf{U}_{k-1}$, the *LU* factorization of $\mathbf{\Gamma}_k=\mathbf{L}_k\mathbf{U}_k$ can be obtained in $\mathcal{O}(k^2)$ operations by solving $\mathbf{L}_{k-1}\mathbf{y}=\mathbf{s}$, $\mathbf{U}_{k-1}^t\mathbf{x}=\mathbf{w}$, and $d=\beta+\mathbf{x}^t\mathbf{y}$. The $\det(\mathbf{\Gamma}_k)$ is then given by

$$\det(\mathbf{\Gamma}_k) = \beta \det(\mathbf{\Gamma}_{k-1}) \tag{A12}$$

Combining Eqs. (A6) and (A10) , $\det(\mathbf{A}_{k+1})$ can be expressed as

$$\det(\mathbf{A}_{k+1}) = (-1)^{k+1}\left(\prod_{j=0}^{k}\gamma_j\right)\det(\mathbf{A}_0)\det(\mathbf{\Gamma}_k) = -\beta\gamma_k \det(\mathbf{A}_k). \tag{A13}$$

Hence, $R_k=-\frac{1}{\beta\gamma_k}$.

The corresponding algorithm is presented in Algorithm 3.

Algorithm 3 Proposed Algorithm during k-th step $(s_p \mapsto s_p')$

```
1: Given $\mathbf{G}_0$, and index arrays $\mathbf{p}$ and $\tilde{\mathbf{p}}$ up to $k-1$ steps
2: Set $\mathbf{p}(k)=p$ and $\tilde{\mathbf{p}}(k)=\tilde{p}$
3: Compute $\gamma_k=(\exp^{-\lambda\sigma(s_p-s_p')}-1)$
4: Set $\mathbf{s}=\mathbf{G}_0(\tilde{\mathbf{p}}(0):\tilde{\mathbf{p}}(k-1),p)$
5: Set $\mathbf{w}^t=\mathbf{G}_0[\tilde{p},\mathbf{p}(0):\mathbf{p}(k-1)]$
6: Set $d=\mathbf{G}_0(\tilde{p},p)-\frac{1}{\gamma_k}$
7: Solve $\mathbf{L}_{k-1}\mathbf{y}=\mathbf{s}$
8: Solve $\mathbf{U}_{k-1}^t\mathbf{x}=\mathbf{w}$
9: Compute $\beta=d-\mathbf{x}^t\mathbf{y}$
10: Compute $R_k=-\frac{1}{\beta\gamma_k}$
11: If the move is accepted, set $\mathbf{L}(k,1:k-1)=\mathbf{x}^t$ and
    $\mathbf{U}(1:k-1,k)=\mathbf{y}$
```

[1] J. Hubbard, Proc. R. Soc. London, Ser. A **276**, 238 (1963).
[2] R. Blankenbecler, D. J. Scalapino, and R. L. Sugar, Phys. Rev. D **24**, 2278 (1981).
[3] R. R. dos Santos, Braz. J. Phys. **33**, No. 1, 36 (2003).
[4] A. Georges, G. Kotliar, W. Krauth, and M. J. Rozenberg, Rev. Mod. Phys. **68**, 13 (1996).
[5] T. Maier, M. Jarrell, T. Pruschke, and M. H. Hettler, Rev. Mod. Phys. **77**, 1027 (2005).
[6] G. Kotliar, S. Y. Savrasov, K. Haule, V. S. Oudovenko, O. Parcollet, and C. A. Marianetti, Rev. Mod. Phys. **78**, 865 (2006).
[7] J. E. Hirsch and R. M. Fye, Phys. Rev. Lett. **56**, 2521 (1986).
[8] A. N. Rubtsov, V. V. Savkin, and A. I. Lichtenstein, Phys. Rev. B **72**, 035122 (2005).
[9] E. Gull, P. Werner, O. Parcollet, and M. Troyer, Europhys. Lett. **82**, 57003 (2008).
[10] M. Jarrell, T. Maier, C. Huscroft and S. Moukouri, Phys. Rev. B **64**,195130 (2001).
[11] J. E. Hirsch, Phys. Rev. B **28**, 4059 (1983).
[12] *Statistical Physics*, edited by K. Binder and D. W. Heermann (Springer Verlag, Berlin, 1992).
[13] G. H. Golub and C. F. van Loan, *Matrix Computations* (The Johns Hopkins University Press, Baltimore, 1996).
[14] G. Alvarez, M. S. Summers, D. E. Maxwell, M. Eisenbach, J. S. Meredith, J. M. Larkin, J. Levesque, T. A. Maier, P. R. C. Kent, E. F. DAzevedo *et al.*, *SC 08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing* (IEEE Press, Piscataway, NJ, 2008), p. 110.
[15] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*, 3rd ed. (Society for Industrial and Applied Mathematics, Philadelphia, 1999).
[16] Our simulations are run on Smoky, an Oak Ridge National Laboratory's Computational Sciences resource. Smoky's current configuration is an 80 node Linux cluster consisting of four quad-core 2.0GHz AMD Opteron processors per node, 32 GB of memory (2 GB/core), a gigabit ethernet network with infiniband interconnect, and access to Spider, the center wide lustre based file system.