2012 IEEE 26th International Parallel and Distributed Processing Symposium

Advancing Large Scale Many-Body QMC Simulations on GPU Accelerated Multicore Systems

Andres Tomas*, Chia-Chen Chang†, Richard Scalettar† and Zhaojun Bai*
*Department of Computer Science, University of California, Davis, CA 95616, USA
{andres,bai}@cs.ucdavis.edu
† Department of Physics, University of California, Davis, CA 95616, USA
cxc639@gmail.com, scalettar@physics.ucdavis.edu

***Abstract*—The Determinant Quantum Monte Carlo (DQMC) method is one of the most powerful approaches for understanding properties of an important class of materials with strongly interacting electrons, including magnets and superconductors. It treats these interactions exactly, but the solution of a system of N electrons must be extrapolated to bulk values. Currently $N \approx 500$ is state-of-the-art. Increasing N is required before DQMC can be used to model newly synthesized materials like functional multilayers.**

DQMC requires millions of linear algebra computations of order N matrices and scales as N^3. DQMC cannot exploit parallel distributed memory computers efficiently due to limited scalability with the small matrix sizes and stringent procedures for numerical stability. Today, the combination of multisocket multicore processors and GPUs provides widely available platforms with new opportunities for DQMC parallelization.

The kernel of DQMC, the calculation of the Green's function, involves long products of matrices. For numerical stability, these products must be computed using graded decompositions generated by the QR decomposition with column pivoting. The high communication overhead of pivoting limits parallel efficiency. In this paper, we propose a novel approach that exploits the progressive graded structure to reduce the communication costs of pivoting. We show that this method preserves the same numerical stability and achieves 70% performance of highly optimized `DGEMM` on a two-socket six-core Intel processor. We have integrated this new method and other parallelization techniques into QUEST, a modern DQMC simulation package. Using 36 hours on this Intel processor, we are able to compute accurately the magnetic properties and Fermi surface of a system of $N = 1024$ electrons. This simulation is almost an order of magnitude more difficult than $N \approx 500$, owing to the N^3 scaling. This increase in system size will allow, for the first time, the computation of the magnetic and transport properties of layered materials with DQMC. In addition, we show preliminary results which further accelerate DQMC simulations by using GPU processors.

***Keywords*-Quantum Monte Carlo; QRP; multicore; GPU**

I. Introduction

The Hubbard Hamiltonian [1] is one of the most important models in condensed matter physics [2], [3], [4]. It provides a theoretical framework for describing electron interactions that are responsible for phenomena such as magnetism, metal-insulator transitions, and high-temperature superconductivity. Quantum Monte Carlo (QMC) simulations have contributed greatly to the understanding of the Hubbard Hamiltonian [5], [6], [7], [8], [9], [10], [11], [12]. Advances in algorithms as well as in hardware have allowed two dimensional lattice sizes up to 24×24 to be simulated [13]. These are large enough to quantify many of the finite size effects which are the primary challenge to QMC, and are of interest to a large class of materials, like the cuprate superconductors, whose fundamental properties are determined by single 2D planes.

An exciting frontier is the application of QMC to the behavior at the interface between materials, that is, where two or more 2D layers come into contact. The interest is driven by advances in materials synthesis [14] which have made possible increasing precision in the creation and characterization of boundaries. These breakthroughs hold the potential for producing new systems with novel, and technologically important, functional properties [15], [16], [17]. To model an interface realistically, six to eight layers must be studied to allow for the most important effects of the boundary to penetrate into the bulk. State-of-the-art QMC simulations of around 500 sites are only barely sufficient for this purpose: a system of eight 8×8 or six 10×10 layers has an aspect ratio for which the extent of each plane is no greater, or only marginally greater, than the dimension perpendicular to the interface.

Thus in order to address the physics of forefront materials science, a further increase in QMC simulation capabilities is required. The capability to simulate eight 12×12 or six 14×14 layers, for example, would allow a very significant enhancement of our ability to model magnetism and conductivity at interfaces. In this paper we report on the development of simulation software using multicore and GPU technology to achieve this aim.

QUantum Electron Simulation Toolbox (QUEST)[1] is a Fortran 90/95 package that implements the Determinant Quantum Monte Carlo (DQMC) method [18], [19] for the Hubbard Hamiltonian [13]. QUEST uses a two-dimensional periodic rectangular lattice as the default geometry. However, the lattice size and physical parameters are very

[1] http://www.cs.ucdavis.edu/∼bai/PETAMAT

1530-2075/12 $26.00 © 2012 IEEE
DOI 10.1109/IPDPS.2012.37

generally configurable through an input file. A great variety of physical measurements, both static and dynamic, can be calculated by QUEST.

Typical matrix sizes in DQMC are not sufficiently large to achieve high parallel performance on a large number of processors. The workload is not enough to compensate the communication overhead. Therefore, DQMC has not been efficiently parallelized on massively distributed memory computer systems. However, current multisocket multicore systems offer shared memory parallelism with low communication overhead. On these systems, linear algebra operations can be efficiently parallelized for small size matrices [20]. In addition, the recent development of GPU computing technology provides an additional parallel computing paradigm which is suitable to these kinds of operations [21], [22]. The combination of multicore processors and GPUs provides an efficient and widely available platform for DQMC parallelization. This paper is focused on the parallel implementation of QUEST on such computer systems. We use optimized BLAS/LAPACK [23] implementations, such as Intel MKL and Nvidia CUBLAS, and develop some OpenMP and CUDA codes for those operations that are not available in these libraries.

Green's function calculation is the computational kernel in QUEST. A Green's function G is a matrix whose rows and columns are labeled by the sites of the lattice. The importance of the Green's function is that it determines the probability for the electron to travel between sites. The Green's function evaluation consists of computing a long product of matrices and matrix inversion. Numerical stability is a critical issue. Traditional algorithms for computing the product of matrices are based on graded decompositions generated by the pivoted QR (QRP) [24]. Although QRP in LAPACK uses level 3 BLAS operations as much as possible, it still requires level 2 BLAS operations for updating the pivot criteria [25]. The performance of the QRP decomposition is only a fraction of the performance of the QR decomposition (without pivoting) in the state-of-the art multicore implementations (see Figure 1).

In this paper, we propose a novel approach that exploits the progressive graded structure in the algorithm to reduce the communication costs of pivoting. This approach replaces most of the pivoted QR decompositions by QR decompositions without pivoting to greatly improve parallel performance, and meanwhile provides the required numerical stability. In addition, we also present an implementation of the Green's function evaluation on a hybrid CPU+GPU platform.

The rest of this paper is organized as follows. Section II gives a brief introduction to DQMC. Section III explains in detail the techniques used in QUEST for Green's function evaluation and for reducing its computational cost. Section IV introduces a novel technique which achieves better parallelism than the previous method on multicore processors for computing the Green's functions. Section V confirms the validity of our implementation via physical observables that can be compared with previous results in the literature. Section VI shows preliminary results using a GPU to further accelerate the Green's function calculation. Concluding remarks are in Section VII.

II. DQMC Simulations

A. The Hubbard Model

The Hubbard model consists of three terms

$$\mathcal{H} = \mathcal{H}_T + \mathcal{H}_V + \mathcal{H}_\mu,$$

where $\mathcal{H}_T$, $\mathcal{H}_V$, and $\mathcal{H}_\mu$ represent kinetic energy, interaction energy, and chemical potential, respectively,

$$\begin{aligned}
\mathcal{H}_T &= -t \sum_{\langle \mathbf{r},\mathbf{r}' \rangle,\sigma} \left(c^\dagger_{\mathbf{r},\sigma} c_{\mathbf{r}',\sigma} + c^\dagger_{\mathbf{r}',\sigma} c_{\mathbf{r},\sigma} \right), \\
\mathcal{H}_V &= U \sum_{\mathbf{r}} n_{\mathbf{r},+} n_{\mathbf{r},-}, \\
\mathcal{H}_\mu &= -\mu \sum_{\mathbf{r}} \left(n_{\mathbf{r},+} + n_{\mathbf{r},-} \right).
\end{aligned}$$

The parameter t is the electron hopping amplitude between nearest-neighbor sites indicated by $\langle \cdot, \cdot \rangle$ in the summation of $\mathcal{H}_T$. $U > 0$ is the strength of repulsive interaction when two electrons with opposite spins occupy the same site. μ controls the chemical potential. The operator $c^\dagger_{\mathbf{r},\sigma}$ ($c_{\mathbf{r},\sigma}$) creates (destroys) an electron on lattice site $\mathbf{r}$ with spin $\sigma \in \{+, -\}$ (up and down). The operator $n_{\mathbf{r},\sigma}$ counts the density of electrons on site $\mathbf{r}$.

At a finite temperature T, the expectation value of a physical observable $\mathcal{O}$, such as the momentum distribution or the spin-spin correlation, is given by the thermal average

$$\langle \mathcal{O} \rangle = \frac{1}{\mathcal{Z}} \text{Tr} \left(\mathcal{O} e^{-\beta \mathcal{H}} \right),$$

where "Tr" is a trace over the Hilbert space of the Hubbard Hamiltonian $\mathcal{H}$, and

$$\mathcal{Z} = \text{Tr} \left(e^{-\beta \mathcal{H}} \right),$$

is the partition function. $\beta = 1/(k_B T)$ is the inverse temperature with k_B being Boltzmann's constant.

Let the inverse temperature be discretized $\beta = L \Delta\tau$, where L denotes the number of inverse temperature slices and $\Delta\tau$ is the time step size. Using the Trotter approximation, the partition function is then written as

$$\begin{aligned}
\mathcal{Z} &= \text{Tr} \left(\prod_{l=1}^{L} e^{-\Delta\tau \mathcal{H}} \right) \\
&= \text{Tr} \left(\prod_{l=1}^{L} e^{-\Delta\tau \mathcal{H}_K} e^{-\Delta\tau \mathcal{H}_V} \right) + O(\Delta\tau^2),
\end{aligned}$$

where $\mathcal{H}_K = \mathcal{H}_T + \mathcal{H}_\mu$ includes the kinetic energy and chemical potential terms which are quadratic in electron

operators. Each of L potential energy terms, which are quartic in electron operators, is decoupled into quadratic form by introducing Hubbard-Stratonovich (HS) fields h, one for each of the lattice sites and inverse temperature discretization intervals where electrons interact. Consequently, the partition function becomes

$$\mathcal{Z} = \sum_{h} |M_+(h)| \, |M_-(h)| \,, \tag{1}$$

where $h = (h_{l,i})$ denotes the HS fields collectively. The matrix $M_\sigma(h)$ for $\sigma \in \{+, -\}$ is defined as

$$M_\sigma(h) = I + B_{L,\sigma}(h_L) B_{L-1,\sigma}(h_{L-1}) \dots B_{1,\sigma}(h_1).$$

I is an identity matrix and for $l = 1, 2, \dots, L$,

$$B_{l,\sigma}(h_l) = e^{\sigma\nu \, \mathrm{diag}(h_{l,1}, h_{l,2}, \dots, h_{l,N})} \, e^{-\Delta\tau K}. \tag{2}$$

The matrix K describes how electrons move among lattice sites, and its diagonal elements contain the chemical potential terms. The quantity $\nu = \cosh^{-1}(e^{U\Delta\tau/2})$ with $U > 0$ parameterizes the strength of electron interactions. The multidimensional summation in (1) is carried out stochastically by Monte Carlo sampling, called the DQMC algorithm, which will be described in next subsection.

B. The DQMC Algorithm

Matrices $M_\sigma(h)$ in the partition function $\mathcal{Z}$ are of dimension N and they involve the product of L matrices. Therefore, the computational complexity of their evaluation is of order $N^3 L$. Currently, DQMC simulations can be done on several hundreds of sites, up to a maximum of $N = 24 \times 24 = 576$. This ability has made the DQMC method suitable for simulating strongly correlated two dimensional materials, since long range correlations (ten or more lattice spacings) can be computed. However, as several layers are considered, the transverse direction has to shrink and this essential long range information will be lost unless algorithmic improvements can be devised.

As described in Section II-A, the DQMC method uses a discrete auxiliary field approach to formulate a computable form of the partition function [18]. The resulting multidimensional summation is then carried out by Monte Carlo sampling. The DQMC method (outlined in Algorithm 1) uses the Metropolis algorithm to find feasible spin configurations via local search on the HS field. Initially, a random configuration of the HS field is given. During the simulation, each element of the HS field is visited, and a new configuration that flips the element's value is proposed. The acceptance of the new configuration is determined by the ratio of the product of determinants before and after flipping. A complete visiting of all LN elements of the HS field is called a sweep.

The DQMC simulation consists of two stages: warmup and sampling. Each stage utilizes the Metropolis algorithm for different purposes. In the warmup stage, the Metropolis algorithm is used to thermalize the field configurations; while in the sampling stage, the physical observables are measured as the field continues to be sampled.

Algorithm 1 DQMC sweep

1) For $l = 1, 2, \cdots, L$
   a) For $i = 1, 2, \cdots, N$
      i) Flip $h'_{l,i} = -h_{l,i}$
      ii) Compute the Metropolis ratio
      $$r_{l,i} = \frac{|M_+(h')| \, |M_-(h')|}{|M_+(h)| \, |M_-(h)|}$$
      iii) Acceptance condition (random number r and $r \sim \mathrm{uniform}[0, 1]$)
      $$h_{l,i} \leftarrow h'_{l,i} \quad \text{if } r_{l,i} \geq r$$
2) Compute physical measurements

The Green's function associated with a configuration h is defined as

$$G^\sigma(h) = M_\sigma(h)^{-1}. \tag{3}$$

Using this function, the Metropolis ratio $r_{l,i}$ in Algorithm 1 can be easily computed thanks to the fact that $M_\sigma(h')$ is a rank-1 update of $M_\sigma(h)$ [26]. Specifically, for $l = 1$, at the spatial site $i = 1$:

$$h'_{1,1} = -h_{1,1}$$

and the Metropolis ratio $r_{1,1}$ is given by

$$r_{1,1} = d_+ d_-,$$

where for $\sigma \in \{+, -\}$,

$$\begin{aligned} d_\sigma &= 1 + \alpha_{1,\sigma}(1 - e_1^T M_\sigma(h)^{-1} e_1) \\ &= 1 + \alpha_{1,\sigma}(1 - G^\sigma_{1,1}(h)), \end{aligned}$$

and

$$\alpha_{1,\sigma} = e^{-2\sigma\nu h_{1,1}} - 1.$$

Therefore, the gist of computing the Metropolis ratio $r_{1,1}$ is to compute the $(1,1)$-element of the Green's function matrix $G^\sigma(h)$ as in (3). If $G^\sigma(h)$ has been computed explicitly in advance, then it is essentially *free*, $O(1)$ operations, to compute the ratio $r_{1,1}$.

If the proposed h' is accepted, then the Green's function is updated by a rank-1 matrix:

$$G^\sigma(h) \leftarrow G^\sigma(h) - \frac{\alpha_{1,\sigma}}{r_{1,1}} u_\sigma w_\sigma^T$$

where $u_\sigma = (I - G^\sigma(h)) e_1$, $w_\sigma = (G^\sigma(h))^T e_1$, and e_1 is the first column of the identity matrix.

Next, at the spatial site $i = 2$:

$$h'_{1,2} = -h_{1,2}.$$

By a similar derivation, we have

$$r_{1,2} = d_+ d_-,$$

where for $\sigma \in \{+, -\}$,

$$d_\sigma = 1 + \alpha_{2,\sigma}(1 - G^\sigma_{1,2}(h)),$$

and

$$\alpha_{2,\sigma} = e^{-2\sigma\nu h_{1,2}} - 1.$$

If the proposed h' is accepted, the Green's function is updated by the rank-1 matrix

$$G^\sigma(h) \leftarrow G^\sigma(h) - \frac{\alpha_{2,\sigma}}{r_{1,2}} u_\sigma w_\sigma^T$$

where $u_\sigma = (I - G^\sigma(h))e_2$ and $w_\sigma = (G^\sigma(h))^T e_2$.

Similarly, for $i = 3, 4, \ldots, N$, we can use the same procedure to compute the Metropolis ratios $r_{1,i}$ and updating the Green's functions if necessary. In QUEST, this update of the Green's functions is delayed to lead to a block rank update instead of individual rank-1 updates [27].

After $i = N$, the Green's function cannot be updated and it must be recomputed from its original formulation. In our previous sequential QUEST implementation, the cost of evaluating these full Green's functions takes roughly 95% of the total simulation time.

III. Green's Function Evaluation

DQMC simulations require a large number of consecutive Green's function evaluations. In this section, the algorithm required for numerical stable evaluation is discussed first. Then, we present techniques implemented in QUEST to reduce its computational cost by exploiting the relation between successive evaluations.

A. Green's Function Evaluation

1) Stratification: In a simplified formulation, the Green's function in (3) is of the form

$$G = (I + B_L B_{L-1} \cdots B_1)^{-1}, \tag{4}$$

where $B_i = B_{i,\sigma}(h_i)$ is defined in (2) and can be recast as

$$B_i = V_i B,$$

$V_i = e^{\sigma\nu \mathrm{diag}(h_{i,1}, h_{i,2}, \ldots, h_{i,N})}$ is a diagonal matrix, and $B = e^{-\Delta\tau K}$ is a matrix exponential which does not change during the simulation.

When L or U is large (that is, low temperatures or strong interactions), the product matrix $B_L B_{L-1} \cdots B_1$ in (4) is extremely ill-conditioned. Several methods have been proposed to stabilize the computation by stratifying the magnitude of elements in the matrix multiplications [28], [29], [19], [24]. All those methods inevitably require the pivoted QR decomposition. The stratification method shown in Algorithm 2 proposed by Loh *et al* [19] is currently used in QUEST to calculate G.

Algorithm 2 Stratification method

1) Compute the pivoted QR: $B_1 = Q_1 R_1 P_1^T$
2) Set $D_1 = \mathrm{diag}(R_1)$ and $T_1 = D_1^{-1} R_1 P_1^T$
3) For $i = 2, 3, \cdots, L$
   a) Compute $C_i = (B_i Q_{i-1}) D_{i-1}$
   b) Compute the pivoted QR: $C_i = Q_i R_i P_i^T$
   c) Set $D_i = \mathrm{diag}(R_i)$ and $T_i = (D_i^{-1} R_i)(P_i^T T_{i-1})$
4) Compute $G = (T_L^{-T} Q_L^T D_b + D_s)^{-T} D_b Q_L^T$

In Algorithm 2, elements of different energy levels, which correspond to different magnitudes of numbers, are stratified by the pivoted QR decomposition. At the last step of the algorithm, D_b and D_s are computed from D_L as

$$D_b(i) = \begin{cases} 1/|D_L(i)| & \text{if } |D_L(i)| > 1 \\ 1 & \text{otherwise} \end{cases}$$

and

$$D_s(i) = \begin{cases} D_L(i) & \text{if } |D_L(i)| \le 1 \\ \mathrm{sgn}(D_L(i)) & \text{otherwise.} \end{cases}$$

The stratification process protects small numbers from being rounded off by mixing with large ones in matrix products. All products in the algorithm involve matrices that have elements of similar magnitude except for the diagonal matrices. The parenthesis in the step 3a instructs to first multiply B_i and Q_{i-1} before multiplying by D_{i-1}. For the typical parameters in DQMC simulations, B_i does not have very large elements and Q_{i-1} is an orthogonal matrix so the product could be computed accurately. Since D_{i-1} is diagonal, the second product is just a column scaling. Therefore, the matrix C_i can be accurately computed. Obviously, we assume that these products can be represented without either overflow or underflow. In the step 3c, the product $T_i = (D_i^{-1} R_i)(P_i^T T_{i-1})$ has also the same property. Here the QR decomposition with pivoting makes the largest element of $D_i^{-1} R_i$ less than one, therefore the product T_i can be stably computed. The splitting of D_L into D_b and D_s in the last step of the algorithm is made for increasing the accuracy of the computation. A numerical stability analysis of the stratification method can be found in [24].

2) Matrix Clustering: In order to reduce the computational cost associated with the pivoted QR in the stratification algorithm, we can work with only $L_k = \lceil \frac{L}{k} \rceil$ matrices by first clustering k of B_i matrices. Specifically, if we define

$$\widehat{B}_i = B_{ik} B_{ik-1} \cdots B_{(i-1)k+2} B_{(i-1)k+1}.$$

for $i = 1, 2, \ldots, L_k$, then the product of L matrices in (4) can be recast as the product of L_k matrices

$$B_L B_{L-1} \cdots B_2 B_1 = \widehat{B}_{L_k} \widehat{B}_{L_{k-1}} \cdots \widehat{B}_1.$$

In this way the number of iterations in the main loop of Algorithm 2 is reduced by a factor k. Usually, a value of

about $k = 10$ gives significant performance boost while maintaining acceptable numerical stability [26].

B. Green's Function Updating

1) Wrapping: As the simulation goes from the inverse temperature slice l to the next $l+1$, the Green's functions can be computed via wrapping. For example, consider the Green's functions from $l = 1$ to $l = 2$ in Algorithm 1, the new Green's function takes the form

$$\widehat{G} = (I + B_1 B_L B_{L-1} \cdots B_2)^{-1}.$$

It is easy to see that $\widehat{G}$ can be computed from the previous G defined in (4) by noting the relation

$$\widehat{G} = B_1 G B_1^{-1}.$$

This is referred to as *wrapping*. One can use the wrapping for a number of times until the computed Green's function loses necessary numerical accuracy. Then the Green's function must be re-computed from scratch by the stratification method (Algorithm 2). A typical value for the number of wrappings is $\ell = 10$.

2) Recycling: When using wrapping and matrix clustering with $k = \ell$ in a full DQMC simulation with thousands of sweeps, the stratification algorithm computes the following sequence of Green's functions

$$\begin{aligned} &(I + \widehat{B}_{L_k} \widehat{B}_{L_k-1} \cdots \widehat{B}_1)^{-1} \\ &(I + \widehat{B}_1 \widehat{B}_{L_k} \widehat{B}_{L_k-1} \cdots \widehat{B}_2)^{-1} \\ &(I + \widehat{B}_2 \widehat{B}_1 \widehat{B}_{L_k} \widehat{B}_{L_k-1} \cdots \widehat{B}_3)^{-1} \\ &\cdots \\ &(I + \widehat{B}_{L_k-1} \widehat{B}_{L_k-2} \cdots \widehat{B}_1 \widehat{B}_{L_k})^{-1} \end{aligned} \tag{5}$$

and then starts over again a new sweep. We note that for the new sweep, only the rightmost matrix cluster of the sequence in (5) are changed in order, the rest of matrix clusters $\widehat{B}_i$ are unchanged. Therefore, to reduce the computational cost, we can store these matrix clusters instead of recomputing them. Typical DQMC simulations require storing less than one hundred matrices of size up to 1024×1024 (about 8MB per matrix). This amount is not significant since we can easily have far more than 1GB of main memory.

IV. Multicore Implementation

The performance bottleneck of the stratification algorithm for Green's function evaluation is the intensive use of the QR decomposition with pivoting (QRP) (line 3b of Algorithm 2). The LAPACK subroutine (`DGEQP3`) uses level 3 BLAS operations for the trailing matrix update during the decomposition. However, the choice of pivots still requires a level 2 operation for computing each Householder reflector [25]. Therefore, `DGEQP3` is not a fully blocked algorithm like the regular QR decomposition (`DGEQRF`). This imposes a limit to the performance on multicore architectures with deep memory hierarchies [30]. Figure 1 shows the performance

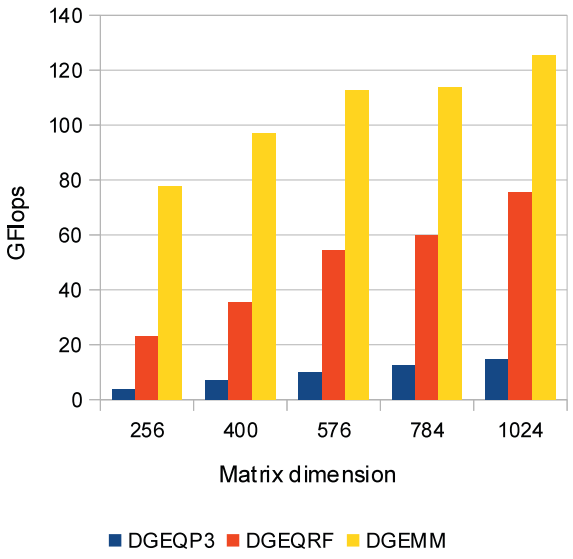

Figure 1. Performance comparison of three LAPACK routines in Intel MKL library on a two-socket six-core Intel Westmere processor.

of three highly used routines in the stratification algorithm on a two-socket six-core Intel Westmere processor.

We observe that the matrix-matrix multiplication (`DGEMM`) has an excellent performance rate even for small matrix sizes. In fact, the `DGEMM` performance rate of matrix size 1024×1024 is close to the maximum rate. The performance rate of the regular QR decomposition (`DGEQRF`) is not as good as `DGEMM` because of the overhead of panel updates in the block QR algorithm for matrices of relative small sizes [31], [32]. But even so, the performance rate of `DGEQRF` is still much better than `DGEQP3`. Therefore, it is highly desirable to modify the stratification algorithm such that the most calls to `DGEQP3` can be replaced by `DGEQRF`.

A. Stratification with Pre-pivoting

A key observation for replacing `DGEQP3` by `DGEQRF` in the stratification algorithm (Algorithm 2) is that as the algorithm iterates, the diagonal elements in D_i are in descending order. It produces an almost column graded matrix C_i. As a result, the QR decomposition with column pivoting of C_i produces the permutation P_i with only very few column interchanges from the initial ordering. Therefore, we propose a variant of the stratification algorithm in Algorithm 3, where a pre-pivoting permutation P_i is computed and applied, and then a regular QR is used. The permutation P_i is computed in the same way as in the QRP algorithm, i.e., the matrix column norms are sorted in descending order. The new algorithm preserves the essential graded structure of C_i although not as strong as in the original stratification algorithm. In

Section IV-C, we will compare the difference of computed Green's functions by two stratification algorithms.

Algorithm 3 Stratification with pre-pivoting method

1) Compute the pivoted QR: $B_1 = Q_1 R_1 P_1^T$
2) Set $D_1 = \mathrm{diag}(R_1)$ and $T_1 = D_1^{-1} R_1 P_1^T$
3) For $i = 2, 3, \cdots, L$
   a) Compute $C_i = (B_i Q_{i-1}) D_{i-1}$
   b) Compute the permutation P_i such as $\widehat{C}_i = C_i P_i$ has decreasing norm columns
   c) Compute a regular QR: $\widehat{C}_i = Q_i R_i$
   d) Set $D_i = \mathrm{diag}(R_i)$ and $T_i = (D_i^{-1} R_i)(P_i^T T_{i-1})$
4) Compute $G = (T_L^{-T} Q_L^T D_b + D_s)^{-T} D_b Q_L^T$

B. OpenMP Parallelization

Algorithm 3 allows us to easily exploit the processing power of multicore systems because most of numerical intensive operations are carried by the QR decompositions and the matrix-matrix products. On the other hand, for the fine-grain operations such as the row and column scaling (lines 3a and 3d) of Algorithm 3) and column norms computation (line 3b), we have to provide our own implementations to benefit from the memory hierarchy parallelism inside the multicore processors for the relatively small matrix sizes in our DQMC simulation. Specifically, for the matrix row and column scaling, by the definition of $B_i = V_i B$, each product by B_i in line 3a also includes a row scaling. These matrix scalings are implemented using OpenMP to distribute the work. To compute column norms for the pre-pivoting, we observe that there is not sufficient workload to achieve good parallel performance from the BLAS routine. Our implementation uses OpenMP to compute several norms simultaneously and obtains better parallel efficiency.

C. Performance Results

In this section, we first check the accuracy of the new stratification algorithm, and then show the performance improvements of Green's function evaluations in QUEST.

In order to check the accuracy of the proposed stratification variant, let us compare the Green's functions computed by Algorithms 2 and 3 for a set of typical values of U used in DQMC simulations. Figure 2 shows the distribution of the relative differences for 1000 Green's function evaluations, sampled from a full DQMC simulation with respect to different value of U. The lattice size is 16×16 with $L = 160$ and $\Delta\tau = 0.2$ ($\beta = 32$). The relative difference is measured by $\|G - \tilde{G}\|_F / \|G\|_F$, where G and $\tilde{G}$ are the Green's functions computed by Algorithms 2 and 3, respectively. We use a box-and-whisker diagram for the distribution to show the minimum, lower quartile (Q1), median (Q2), upper quartile (Q3), and maximum differences. These results indicate that most differences are below 10^{-12}. Furthermore, the value of U does not have a significant impact on the accuracy of the Green's functions evaluated by the new stratification algorithm.

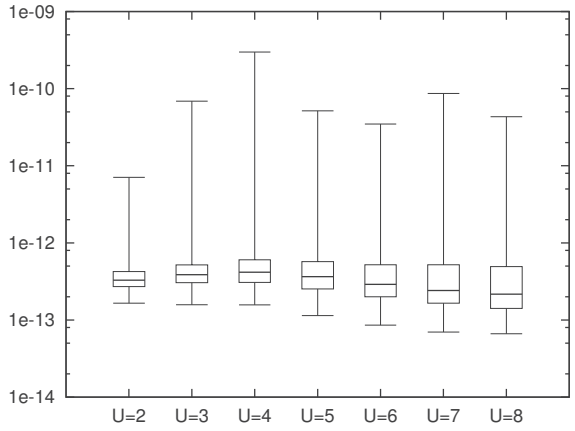

Figure 2. Distribution of the relative differences between the original and proposed algorithms for evaluating 1000 Green's functions in DQMC simulations.

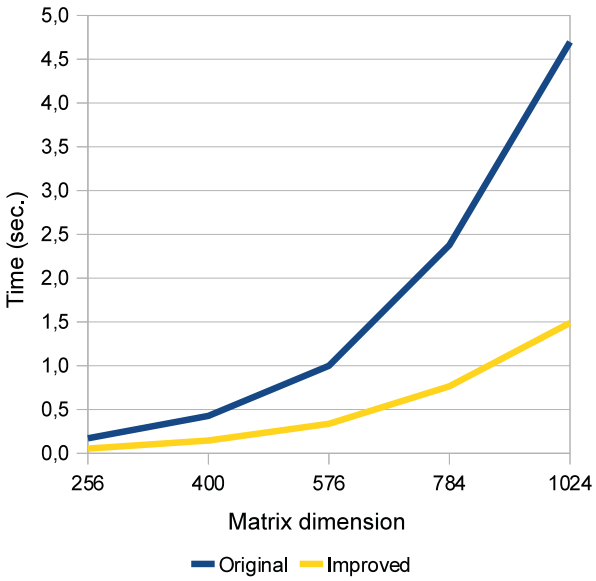

Figure 3. Average time required for a Green's function evaluation on a two-socket six-core Intel Westmere processor.

Next let us show the performance improvements of the Green's function evaluation in QUEST. Figures 3 and 4 shows the CPU elapsed time and the GFlops rate for evaluating G with different matrix dimensions (number of sites N) and $L = 160$. These tests were run using a two-socket six-core Intel Westmere processor. The performance reported is the average in a full DQMC simulation that takes 1000 warming and 2000 measurements sweeps (Algorithm 1).

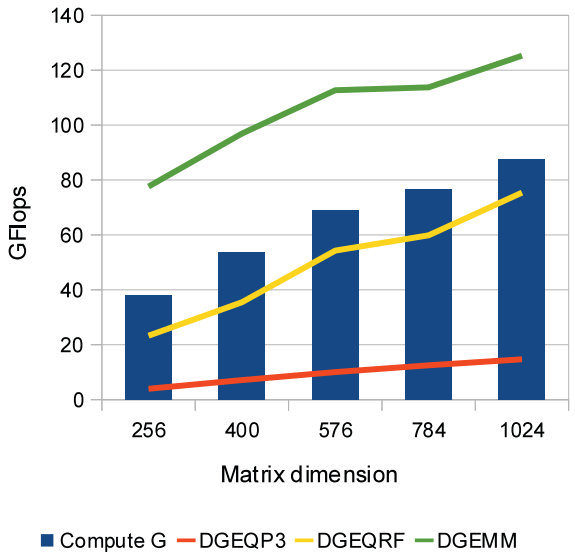

Figure 4. Green's function evaluation performance in GFlops on a two-socket six-core Intel Westmere processor.

From Figure 3, we see that the execution time is up to three times faster due to the performance enhancement techniques presented in this paper, namely, stratification with pre-pivoting and reuse of matrix clustering with $k = \ell = 10$.

The cost of Algorithm 3 is $O(N^3 L)$. There are several operations like matrix scaling by a diagonal and norm computations (level 2 BLAS operations) with total cost $O(N^2 L)$. As N is relatively small, the impact of these operations is not negligible. Taking into account this effect, the performance of the improved Green's function evaluation is quite good, roughly 70% of the `DGEMM` GFlops rate and even better than the QR decomposition (`DGEQRF`) as shown in Figure 4.

V. Physical Measurements

To demonstrate the new capability of multicore-based QUEST, we show the results of two physical measurements from full DQMC simulations with 1000 warming and 2000 measurement sweeps.

A. Physical Measurements

Two important physical observables of the Hubbard model are the momentum distribution

$$\langle n_{\mathbf{k},\sigma} \rangle$$

and the z-component spin-spin correlation function

$$C_{zz}(\mathbf{r}) = \frac{1}{N} \sum_{\mathbf{r}'} \langle (n_{\mathbf{r}+\mathbf{r}',+} - n_{\mathbf{r}+\mathbf{r}',-}) (n_{\mathbf{r}',+} - n_{\mathbf{r}',-}) \rangle.$$

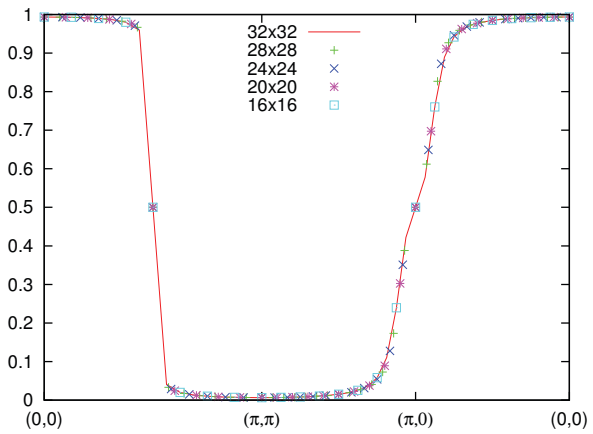

Figure 5. Mean momentum distribution $\langle n_{\mathbf{k}} \rangle$ of the two-dimensional Hubbard model at average density $\rho = 1$, interaction strength $U = 2$, and inverse temperature $\beta = 32$. $\langle n_{\mathbf{k}} \rangle$ is plotted along the momentum space symmetry line $(0,0) \to (\pi,\pi) \to (\pi,0) \to (0,0)$.

In these equations, $n_{\mathbf{k},\sigma}$ and $n_{\mathbf{r},\sigma}$ are electron density operators in momentum and real space respectively, and $\sigma \in \{+,-\}$. Expectation value of the real space density operator $n_{\mathbf{r},\sigma}$ can be extracted from the diagonal terms of the $N \times N$ Green's function matrix

$$\langle n_{\mathbf{r},\sigma} \rangle = G^{\sigma}(\mathbf{r},\mathbf{r}).$$

Its momentum space counterpart $\langle n_{\mathbf{k},\sigma} \rangle$ is obtained from the Fourier transformation of the full Green's function matrix

$$\langle n_{\mathbf{k},\sigma} \rangle = \frac{1}{N} \sum_{\mathbf{r},\mathbf{r}'} e^{i\mathbf{k}\cdot(\mathbf{r}-\mathbf{r}')} G^{\sigma}(\mathbf{r},\mathbf{r}').$$

The momentum distribution $\langle n_{\mathbf{k},\sigma} \rangle$ is an important quantity because it provides the information of the Fermi surface (FS) and the renormalization factor (discontinuity at the FS). Both properties are fundamental quantities in the so-called Fermi liquid theory – one of the most important theoretical paradigms in condensed matter physics and materials science. In Figure 5, the mean momentum distribution (averaged over two spin components) is plotted along the symmetry line in the momentum space for several lattice sizes. At the interaction strength $U = 2$, a sharp FS can be identified near the middle of the segment $(0,0) \to (\pi,\pi)$. Results for $N \leq 576$ are in agreement with published results [13]. Most importantly, $\langle n_{\mathbf{k}} \rangle$ obtained on the 32×32 lattice provides a much better estimation of the renormalization factor due to its excellent spatial resolution in the momentum space.

To further illustrate the benefit gained from the large-scale 32×32 lattice simulation, the color contour plot of the mean momentum distribution is shown in Figure 6 for two different lattice sizes. It is clear that result obtained on the 32×32 lattice reveals much more detail than the 12×12 data.

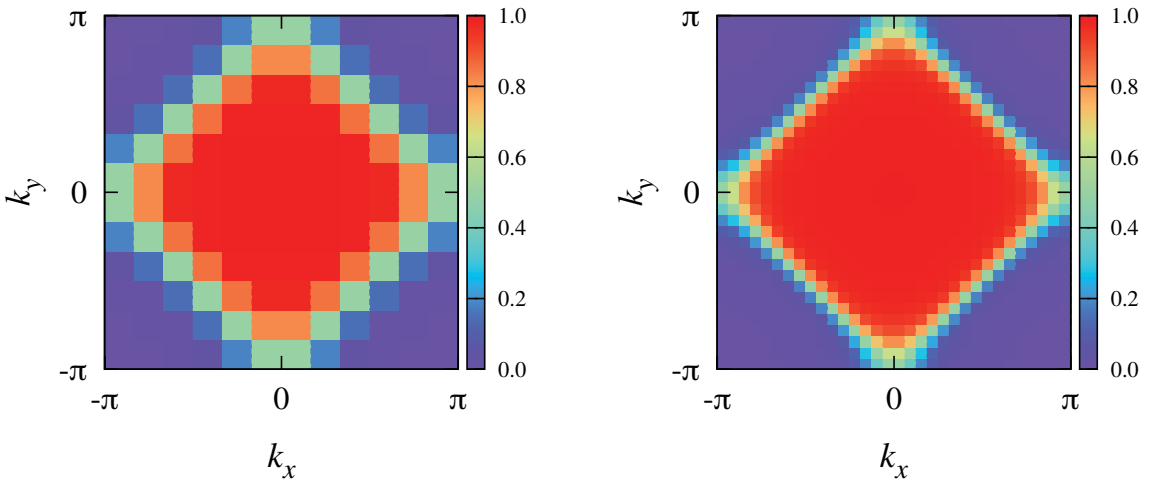

Figure 6. Color contour plot of the mean momentum distribution $\langle n_{\mathbf{k}} \rangle$ of the two-dimensional Hubbard model on 12×12 (left) and 32×32 (right) lattices. Simulation parameters are the same as Figure 5.

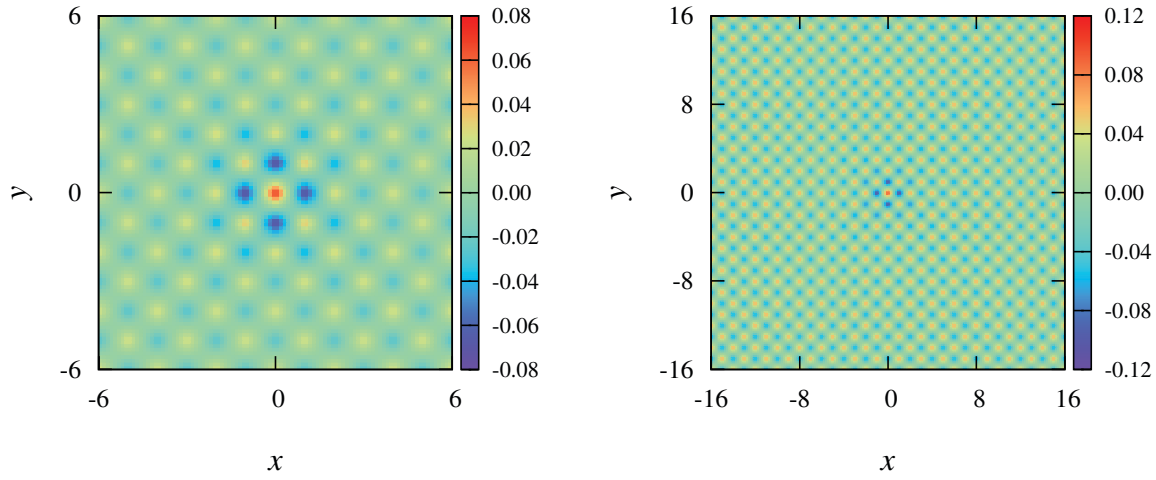

Figure 7. z-component spin-spin correlation $C_{zz}(\mathbf{r})$ on 12×12 (left) and 32×32 lattices with average density $\rho = 1$, interaction strength $U = 2$, and inverse temperature $\beta = 32$.

Next we examine the z-component spin-spin correlation function $C_{zz}(\mathbf{r})$ which is often used to measure the magnetic structure in the Hubbard model. In the simulated case, namely average density $\rho = 1$, interaction strength $U = 2$, and inverse temperature $\beta = 32$, the Hubbard model exhibits an antiferromagnetic (AF) order where electron spin-spin correlation shows a chessboard pattern. This is demonstrated in Figure 7. However, in order to determine whether there is a true magnetic order in the bulk limit $N \to \infty$, the correlation function at the longest distance $C_{zz}(L_x/2, L_y/2)$ will need to be measured on different lattice sizes. The results are then extrapolated to the $N \to \infty$ limit to determine the existence of the magnetic structure in the bulk limit. While both panels in Figure 7 show AF order, it is clear that results obtained on large lattices provide a better estimation of the asymptotic behavior of $C_{zz}(L_x/2, L_y/2)$.

B. CPU Elapsed Time and Profile

Figure 8 shows the CPU elapsed time for the full DQMC simulation with 1000 warming and 2000 measurement sweeps. The line in the plot is the nominal execution time predicted by the cost $O(N^3 L)$ of DQMC using the 256 sites

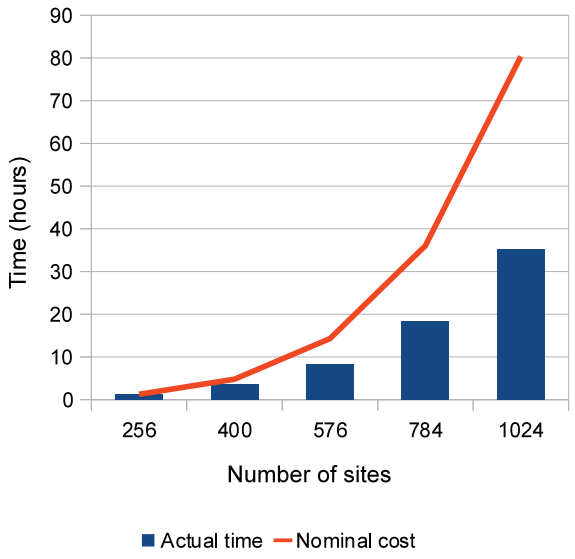

Figure 8. Actual time and nominal prediction (based on N^3 scaling) for a whole DQMC simulation with different number of sites.

	Number of sites				
	256	400	576	784	1024
Delayed rank-1 update	14.2%	16.5%	16.7%	14.9%	13.9%
Stratification	48.5%	45.5%	44.1%	44.5%	44.2%
Clustering	8.4%	9.1%	9.7%	11.3%	12.0%
Wrapping	8.8%	9.4%	10.2%	11.5%	11.9%
Physical meas.	20.0%	19.4%	19.2%	17.9%	18.0%

Table I
EXECUTION TIME IN PERCENTAGE OF THE DIFFERENT STEPS IN A QUEST SIMULATION.

simulation as a reference. The actual CPU elapsed time for 256 sites is 1.25 hours but for 1024 site it is only 28 times as much (35.3 hours). This is better than predicted by the asymptotical cost, where a simulation with $4N$ sites should be $4^3 = 64$ times slower than a simulation with N sites. Note that the simple complexity $O(N^3L)$ does not take into account that the parallel efficiency of linear algebra operation improves as the matrix size grows as long as it still fits in cache memory. For the number of sites (matrix sizes) reported here, this parallel performance improvement is sufficiently significant to partially compensate for the $O(N^3L)$ asymptotical cost of DQMC simulations.

Table I shows the relative computation time for the major steps in a full DQMC simulation. As we can see that the computational time of Green's function evaluations (i.e., stratification, clustering and wrapping) is about 65% of overall simulation time on the multicore system, and is reduced reduced from 95% in the previous sequential QUEST implementation.

VI. GPU ACCELERATION

General-Purpose Graphics Processing Units currently offer more computational power than multicore processors at a lower cost. Development environment for GPUs is evolving towards greater simplicity of programming on each new hardware generation, although it is still hard to program and more difficult to achieve high efficiency than multicore CPUs. An easy way of using GPUs for numerical applications is to use optimized libraries, such as Nvidia CUBLAS [33]. These libraries follow the standard BLAS notion and provide standard interfaces for essential linear algebra operations while hiding details of implementations from users. These details include non-trivial aspects like work distribution among processors and memory access patterns. In this section we present some preliminary results on using a GPU to accelerate the matrix clustering operation (Section III-A2) and Green's function wrapping operation (Section III-B1).

A. Matrix Clustering

Algorithm 4 shows an implementation of the chain product of several $B_i = V_i B$ matrices for the matrix clustering discussed in Section III-B1. A common bottleneck of using GPU is on data transfer between CPU and GPU memories. In DQMC, the matrix B is fixed and it is computed and stored at the start of simulation. The diagonal matrices V_i change and needed to be copied to the graphics memory each time. The resulting matrix A must be copied back to the main memory. The transactions of $NL + N^2$ floating point values are relatively small and are not relevant compared to the whole execution time.

Algorithm 4 Compute $A = B_{i+1} \cdots B_{i+k}$ using CUBLAS

Send $V_{i:i+k}$ to GPU	`cublasSetMatrix`
$T \leftarrow B$	`cublasDcopy`
for $j = 1, 2, \ldots, n$	
$\quad A_{j,1:n} \leftarrow V_{i,j} \times T_{j,1:n}$	`cublasDscal`
end	
for $l = 2, 3, \ldots, k$	
$\quad T \leftarrow B \times A$	`cublasDgemm`
$\quad$ **for** $j = 1, 2, \ldots, n$	
$\quad\quad T_{j,1:n} \leftarrow V_{i+l,j} \times T_{j,1:n}$	`cublasDscal`
$\quad$ **end**	
$\quad A \leftarrow T$	`cublasDcopy`
end	
Send A to CPU	`cublasGetMatrix`

The computation of each $B_i = V_i B$ in Algorithm 4 is made by a copy of the B matrix and repeated calls to the vector scaling routine for each row. This trivial implementation is not quite efficient because each level 1 BLAS routine is not able to achieve the full performance of the

GPU. The latest version of CUBLAS provides a method to simultaneously execute several kernels, allowing to scale all rows of a matrix in parallel. However, the parallel speedup obtained in this case does not compensate the overhead of kernel management. More important is that the memory access pattern is row by row which does not exploit the coalescing memory access features of the graphics memory. Algorithm 5 is an alternative CUDA kernel for computing the product $B_i = V_i B$. Here each thread of the GPU computes the scaling of one matrix row. This guarantees that there are sufficient threads to keep the multiple computing elements occupied and all threads do exactly the same operations in the same order. Both conditions are critical to get good performance. Also, consecutive threads read and write in consecutive memory positions. This technique supports so called coalesced accesses to improve memory bandwidth. Moreover, the number of memory reads is minimized by storing the V_i value in local memory for each thread. Like level 1 BLAS routines, this procedure does not exploit the full computational speed of the processor because is it limited by memory access, but it is optimal in terms of memory bandwidth. Last but not least, this procedure allows us to avoid the matrix copy as in Algorithm 4.

Algorithm 5 CUDA kernel for $B_i = \mathrm{diag}(V) \times B$

$k = \mathrm{blockIdx}.x \times \mathrm{blockDim}.x + \mathrm{threadIdx}.x$
if $k < n$
  $t \leftarrow V_k$
  for $j = 1, 2, \ldots, n$
    $B_{ik,j} \leftarrow t \times B_{k,j}$
  end
end

B. Wrapping

The GPU can also be used for accelerating the wrapping operation (Section III-B1). Algorithm 6 is an implementation of $\widehat{G} = B_i G B_i^{-1} = V_i B G B^{-1} V_i^{-1}$. As before, B and B^{-1} can be computed and stored in the memory of GPU at the start of computation. The scaling by the diagonal V_i is difficult to compute efficiently. Algorithm 7 is the CUDA kernel for a more efficient way of computing the scaling. This implementation is based on Algorithm 5 with the addition of the column scaling factor u. This value is different for each element inside the loop, giving a non-coalesced memory access for each iteration. However, as all threads read simultaneously the same position a texture cache can be used to increase memory bandwidth.

C. Preliminary Performance Results

In this subsection, we report preliminary performance results of the Green's function evaluations (Algorithm 3) on a hybrid CPU+GPU system. The tests were run on one node of Carver at NERSC, which has a two-socket four-core Intel Nehalem processor and a Nvidia Tesla C2050 GPU processor with 448 computing elements. Both Intel MKL and Nvidia CUBLAS libraries are available on Carver.

Algorithm 6 Compute $G \leftarrow B_i G B_i^{-1}$ using CUBLAS

Send G to GPU	`cublasSetMatrix`
Send V_i to GPU	`cublasSetVector`
$T \leftarrow B \times G$	`cublasDgemm`
$G \leftarrow T \times B^{-1}$	`cublasDgemm`
for $i = 1, 2, \ldots, n$	
$G_{i,1:n} \leftarrow V_i \times G_{i,1:n}$	`cublasDscal`
end	
for $j = 1, 2, \ldots, n$	
$G_{1:n,j} \leftarrow A_{1:n,j} / G_i$	`cublasDscal`
end	
Send G to CPU	`cublasGetMatrix`

Algorithm 7 CUDA kernel for computing $G \leftarrow \mathrm{diag}(V) \times G \times \mathrm{diag}(V)^{-1}$

$k = \mathrm{blockIdx}.x \times \mathrm{blockDim}.x + \mathrm{threadIdx}.x$
if $i < n$
  $t \leftarrow V_k$
  for $j = 1, 2, \ldots, n$
    $u \leftarrow V_j$ (via texture)
    $A_{i,j} \leftarrow t \times A_{i,j} / u$
  end
end

Figure 9 shows Gflops rates of GPU implementations for matrix clustering and wrapping, including memory transfer times. As we can see, the CUDA implementation of matrix clustering is close to GPU `DGEMM` performance. This is partially due to the fact that k matrix-matrix products are performed on GPU with only one memory transfer. On the other hand, the GPU implementation of wrapping only computes two matrix-matrix products for each data transfer and does not achieve the same level of performance as the matrix clustering. However, the performance is much better than the CPU `DGEMM` and improves with the matrix dimension.

Figure 10 shows the performance of a hybrid CPU+GPU implementation for computing the whole Green's functions ($L = 160$). The hybrid implementation combines all performance enhancement strategies discussed in this paper, namely stratification with pre-pivoting, matrix clustering, wrapping and reuse of matrix clusters with $k = \ell = 10$. The reported performance is the average of several runs during a simulation. The performance results illustrate great promises of QUEST on the multicore CPU system with GPU

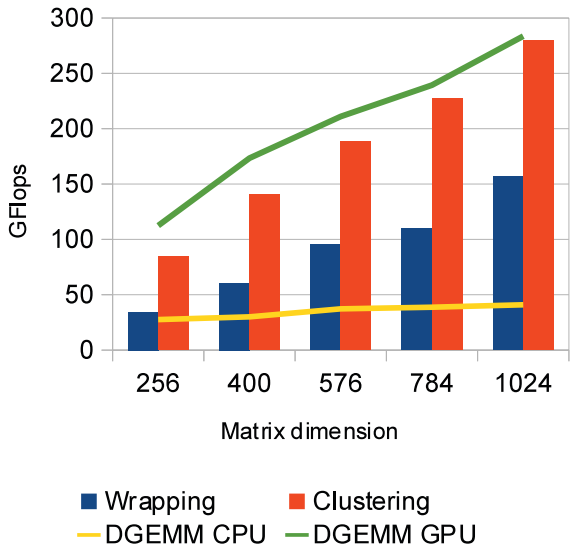

Figure 9. Performance of matrix clustering (Algorithm 4) and wrapping (Algorithm 6) on GPU.

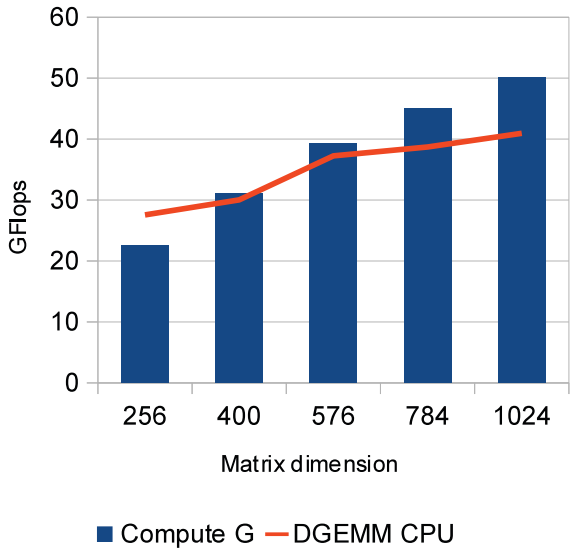

Figure 10. Performance of the Green's function evaluation on the hybrid CPU+GPU system.

acceleration. Our future research direction is to implement most of the stratification procedure (Algorithm 3) on the GPU using the recent advances for the QR decomposition on these systems [34], [35].

VII. Concluding remarks

We anticipate that the combination of multicore processors and GPU accelerators will allow to increase the lattice sizes to a level never being tried before for DQMC simulations. These larger systems will allow Quantum Monte Carlo simulations, which have been essential to the understanding of two dimensional systems, to be applied to a new and very important set of multi-layer materials.

Acknowledgments. This work was supported in part by the U.S. DOE SciDAC grant DOE-DE-FC0206ER25793 and NSF grant PHY1005502. This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. DOE under Contract No. DE-AC02-05CH11231.

References

[1] J. Hubbard, "Electron correlations in narrow energy bands," *Proc. R. Soc. London, Ser A*, vol. 276, p. 283, 1963.

[2] D. J. Scalapino and S. R. White, "Numerical results for the Hubbard model: Implications for the high T_c pairing mechanism," *Found. Phys.*, vol. 31, p. 27, 2001.

[3] D. J. Scalapino, "Numerical studies of the 2D Hubbard model," in *Handbook of High Temperature Superconductivity*, J. R. Schrieffer and J. S. Brooks, Eds. Springer, 2007, ch. 13.

[4] P. W. Anderson, "The resonating valence bond state in la_2cuo_4 and superconductivity," *Science*, vol. 235, p. 1196, 1987.

[5] J. E. Hirsch and S. Tang, "Antiferromagnetism in the two-dimensional Hubbard model," *Phys. Rev. Lett.*, vol. 62, pp. 591–594, Jan 1989.

[6] S. R. White, D. J. Scalapino, R. L. Sugar, E. Y. Loh, J. E. Gubernatis, and R. T. Scalettar, "Numerical study of the two-dimensional Hubbard model," *Phys. Rev. B*, vol. 40, pp. 506–516, Jul 1989.

[7] L. Chen, C. Bourbonnais, T. Li, and A.-M. S. Tremblay, "Magnetic properties of the two-dimensional Hubbard model," *Phys. Rev. Lett.*, vol. 66, pp. 369–372, Jan 1991.

[8] M. Jarrell, "Hubbard model in infinite dimensions: A quantum Monte Carlo study," *Phys. Rev. Lett.*, vol. 69, pp. 168–171, Jul 1992.

[9] R. Preuss, F. Assaad, A. Muramatsu, and W. Hanke, *The Hubbard Model: its Physics and Mathematical Physics*, D. Baeriswyl, D. K. Campbell, J. M. Carmelo, F. Guinea, and E. Louis, Eds. Springer, New York, 1995.

[10] S. Zhang and H. Krakauer, "Quantum Monte Carlo method using phase-free random walks with slater determinants," *Phys. Rev. Lett.*, vol. 90, p. 136401, Apr 2003.

[11] A. Georges, G. Kotliar, W. Krauth, and M. J. Rozenberg, "Dynamical mean-field theory of strongly correlated fermion systems and the limit of infinite dimensions," *Rev. Mod. Phys.*, vol. 68, pp. 13–125, Jan 1996.

[12] T. Maier, M. Jarrell, T. Pruschke, and M. H. Hettler, "Quantum cluster theories," *Rev. Mod. Phys.*, vol. 77, pp. 1027–1080, Oct 2005.

[13] C. N. Varney, C.-R. Lee, Z. J. Bai, S. Chiesa, M. Jarrell, and R. T. Scalettar, "Quantum Monte Carlo study of the two-dimensional fermion Hubbard model," *Phys. Rev. B*, vol. 80, no. 7, p. 075116, Aug 2009.

[14] S. B. Ogale and J. Mannhart, "Interfaces in materials with correlated electron systems," in *Thin Films and Heterostructures for Oxide Electronics*, ser. Multifunctional Thin Film Series, O. Auciello and R. Ramesh, Eds. Springer US, 2005, pp. 251–278.

[15] J. Mannhart and D. G. Schlom, "Oxide interfaces – an opportunity for electronics," *Science*, vol. 327, no. 5973, pp. 1607–1611, 2010.

[16] J. K. Freericks, *Transport in Multilayered Nanostructures: The Dynamical Mean Field Theory Approach*. Imperial College Press, London, 2006.

[17] S. B. Ogale and A. Millis, "Electronic reconstruction at surfaces and interfaces of correlated electron materials," in *Thin Films and Heterostructures for Oxide Electronics*, ser. Multifunctional Thin Film Series, O. Auciello and R. Ramesh, Eds. Springer US, 2005, pp. 279–297.

[18] R. Blankenbecler, D. J. Scalapino, and R. L. Sugar, "Monte Carlo calculations of coupled boson-fermion systems. i," *Phys. Rev. D*, vol. 24, pp. 2278–2286, Oct 1981.

[19] E. Y. Loh, Jr. and J. E. Gubernatis, *Stable Numerical Simulations of Models of Interacting Electrons in Condensed Matter Physics*. Elsevier Sicence Publishers, 1992, ch. 4.

[20] A. Buttari, J. Langou, J. Kurzak, and J. Dongarra, "A class of parallel tiled linear algebra algorithms for multicore architectures," *Parallel Computing*, vol. 35, no. 1, pp. 38–53, 2009.

[21] S. Tomov, J. Dongarra, and M. Baboulin, "Towards dense linear algebra for hybrid GPU accelerated manycore systems," *Parallel Computing*, vol. 36, no. 5-6, pp. 232–240, 2010.

[22] S. Tomov, R. Nath, H. Ltaief, and J. Dongarra, "Dense linear algebra solvers for multicore with GPU accelerators," in *International Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW)*, Apr 2010, pp. 1–8.

[23] 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. SIAM, Philadelphia, 1999.

[24] Z. Bai, C.-R. Lee, R.-C. Li, and S. Xu, "Stable solutions of linear systems involving long chain of matrix multiplications," *Linear Algebra and its Applications*, vol. 435, no. 3, pp. 659–673, 2010.

[25] G. Quintana-Ortí, X. Sun, and C. H. Bischof, "A BLAS-3 version of the QR factorization with column pivoting," *SIAM Journal on Scientific Computing*, vol. 19, no. 5, pp. 1486–1494, 1998.

[26] Z. Bai, W. Chen, R. Scalettar, and I. Yamazaki, "Numerical methods for quantum Monte Carlo simulations of the Hubbard model," in *Multi-Scale Phenomena in Complex Fluids*, T. Y. H. *et al*, Ed. Higher Education Press, 2009, pp. 1–110.

[27] M. Jarrell, private comunication.

[28] G. Sugiyama and S. Koonin, "Auxiliary field Monte-Carlo for quantum many-body ground states," *Annals of Physics*, vol. 168, no. 1, pp. 1 – 26, 1986.

[29] S. Sorella, S. Baroni, R. Car, and M. Parrinello, "A novel technique for the simulation of interacting fermion systems," *EPL (Europhysics Letters)*, vol. 8, no. 7, p. 663, 1989.

[30] Z. Drmač and K. Veselić, "New fast and accurate Jacobi SVD algorithm. i," *SIAM J. Matrix Anal. Appl.*, vol. 29, pp. 1322–1342, January 2008.

[31] B. Hadri, H. Ltaief, E. Agullo, and J. Dongarra, "Enhancing parallelism of tile QR factorization for multicore architectures," *Submitted to Transaction on Parallel and Distributed Systems*.

[32] E. Agullo, J. Dongarra, R. Nath, and S. Tomov, "A fully empirical autotuned dense QR factorization for multicore architectures," in *Proceedings of the 17th international conference on Parallel processing - Volume Part II*, ser. Euro-Par'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 194–205.

[33] "CUBLAS Library", NVIDIA Corporation, *avaliable at* http://developer.nvidia.com/cublas

[34] E. Agullo, C. Augonnet, J. Dongarra, M. Faverge, H. Ltaief, S. Thibault, and S. Tomov, "QR factorization on a multicore node enhanced with multiple GPU accelerators," in *International Parallel Distributed Processing Symposium (IPDPS)*, May 2011, pp. 932–943.

[35] M. Anderson, G. Ballard, J. Demmel, and K. Keutzer, "Communication-avoiding QR decomposition for GPUs," *International Parallel Distributed Processing Symposium (IPDPS)*, pp. 48–58, 2011.