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

Andrés Tomás*, Chia-Chen Chang†, Richard Scalettar‡ and Zhaojun Bai§
*Dept. of Computer Science
University of California, Davis, CA 95616
Email: andres@cs.ucdavis.edu
†Dept. of Physics
University of California, Davis, CA 95616
Email: cxc639@gmail.com
‡Dept. of Physics
University of California, Davis, CA 95616
Email: scalettar@physics.ucdavis.edu
§Dept. of Computer Science
University of California, Davis, CA 95616
Email: bai@cs.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 and increasing N is required before DQMC can be used to model newly synthesized materials like functional multilayers.**

The DQMC requires millions of linear algebra computations of order N matrices and scales as N^3. Over years, researchers have come to realize that DQMC cannot exploit distributed memory parallel computers efficiently due to limited scalability with the small matrix sizes and stringent procedures for numerical stability. Today, the combination of multi-socket multi-core processors and GPUs provide widely available platforms and opportunities for DQMC parallelization.

The kernel of DQMC is the Green's function calculations, which involve long products of matrices. These products must be computed using a graded decomposition generated by the pivoted QR. 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 the same Intel processor, we are able to compute accurately the magnetic properties and Fermi surface of a system of $N = 1024$ electrons, which is almost an order of magnitude more difficult than $N \approx 500$, owing to the N^3 scaling. We will also show preliminary results which further accelerate DQMC simulations by using GPU processors. This increase in system will size allow, for the first time, the computation of the magnetic and transport properties of layered materials with DQMC.

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

I. Introduction

The Hubbard model [1] is one of the most important models in condensed matter physics [2], [3] and material sciences [4]. It provides a theoretical framework for describing electron interactions that are responsible for several fascinating phenomena such as magnetism and high-temperature superconductivity. Quantum Monte Carlo (QMC) simulations have contributed greatly to the understanding of magnetic correlations and metal-insulator transitions of this Hamiltonian [5], [6], [7], [8], [9], [10], [11], [12]. Advances in algorithms as well as in hardware have allowed lattice sizes up to 24x24 to be simulated [13]. These are large enough to address the finite size effects which are the primary challenge to QMC.

Attention is now turning to exploration of the behavior at the interface between materials. Advances in materials synthesis [14] 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]. However, this endeavor also raises new challenges to QMC simulations. In order to model an interface, one needs minimally to couple two 2D layers. More 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 8x8 layers has an aspect ratio for which the extent of each plane is 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. In this paper we report on the development of simulation software using recent multicore and GPU technology which achieves 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]. QUEST have been extensively used to study magnetism, metal-insulator transition and superconductivity in the Hubbard Hamiltonian. QUEST uses a two-dimensional periodic rectangular lattice as the default geometry in the simulation. The lattice size and physical parameters are configurable through an input file and several physical measurements for the Hubbard model can be calculated by the program.

Matrix sizes used in DQMC are not sufficiently big to achieve good parallel performance with a large number of processors. This workload is not enough to compensate the communication overhead even with a few processors. Therefore, DQMC cannot be efficiently parallelized in distributed memory computers. However, current multisocket multicore computers offer shared memory parallelism with low communication overhead. In these computers, linear algebra operations can be efficiently parallelized even for the small matrices involved [20]. Also, recent GPUs provide a similar parallel paradigm which is suitable to these kind of operations [21], [22]. Therefore, the combination of multicore processors and a GPU provide an efficient and widely available platform for DQMC parallelization. This paper is focused on the parallel implementation of QUEST in such computers. This parallelization uses optimized BLAS/LAPACK [23] implementations, such as Intel MKL and Nvidia CUBLAS, and custom OpenMP/CUDA code for some operations not available in these libraries.

Green's function calculation is the main kernel in QUEST. The Green's function G_{ij} 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 i and j. Specifically, this evaluation consists of computing the inverse of a long product of matrices and numerical stability is a critical issue. Traditional algorithms for computing a decomposition of this product of matrices are based on graded decompositions generated by the pivoted QR (QRP) [24] that guarantees the numerical stability of the procedure.

Although QRP in the LAPACK library has been extended to use BLAS level 3 operations as much as possible, still requires a BLAS level 2 operation for updating the pivot criteria [25]. Our results show, for the matrix sizes used in DQMC simulations, that the QRP decomposition is not as efficient as the standard QR in state of the art multicore implementations of LAPACK. 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 regular QR decompositions, providing equivalent numerical stability and greatly improved parallel performance.

The rest of this paper is organized as follows. In section II, we present a brief introduction to DQMC simulations and show the relevance of Green's function evaluations. Section III explains in detail the techniques used in QUEST for Green's function evaluation and for reducing its computational cost. In section IV we introduce a novel technique which achieves better parallel performance than the traditional approach in multicore processors. Section V confirms the validity of our implementation by computing some simulations that can be compared with previous results in the literature. Our results show that QUEST is as robust as previous DQMC implementations but faster allowing to tackle simulations with bigger number of sites. Finally, section VI shows preliminary results using a GPU to further accelerate DQMC simulations and the paper is summarized 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 $\mathcal{H}_\mu$ are kinetic energy, interaction energy, and chemical potential terms respectively. At a finite temperature T, the expectation value of a physical observable $\mathcal{O}$ in the Hubbard model, for example the momentum distribution or the spin-spin correlation, is given by the thermal average

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

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

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

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

In a numerical simulation, the Hubbard model is put on a lattice of $N = L_x \times L_y$ sites. Next, the inverse temperature is discretized $\beta = L\Delta\tau$, where L denotes the number of inverse temperature slices and $\Delta\tau$ is the time step size. The partition function is then written as

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

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

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

operators. Each of the L potential energy terms, which is quartic in electron operators, is decoupled into quadratic form by introducing N binary Hubbard-Stratonovich (HS) fields $h_{l,i} = \pm 1$, one for each of the lattice sites where electrons interact. After these steps, the trace can be evaluated analytically and the partition function becomes

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

with h denotes the $N \times L$ HS fields collectively. The matrix $M_\sigma(h)$ ($\sigma = \pm$) is defined as

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

where I is an identity matrix, and

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

The $N \times N$ matrix K describes how electrons move among lattice sites, and its diagonal elements contain the chemical potential terms. The parameter $\nu = \cosh^{-1}(e^{U\Delta\tau/2})$ with $U > 0$ parameterizes the strength of electron interactions. The multidimensional summation in Eq. (1) is carried out stochastically by Monte Carlo sampling which will be described in the next subsection. For a more detailed derivation of the partition function and the DQMC algorithm, we refer the reader to the article [18].

B. DQMC simulations

In the DQMC simulation, matrices are of the dimension N. They involve the product of L matrices. The computational complexity of the sequential code is of order $N^3 L$. Currently, DQMC simulations can be done on several hundreds of sites, up to a maximum of around $N = 500$. This ability makes the DQMC method particularly 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 the previous subsection, 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 by Monte Carlo sampling. The DQMC method implements 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 (Algorithm 1).

Algorithm 1 DQMC sweep

1) $l = 1$, $i = 1$
2) Flip $h'_{l,i} = -h_{l,i}$
3) Compute the Metropolis ratio
$$r_{l,i} = \frac{|M_+(h')| \, |M_-(h')|}{|M_+(h)| \, |M_-(h)|}$$
4) Acceptance condition (random number r)
$$h_{l,i} \leftarrow h'_{l,i} \text{ if } r \le r_{l,i}$$
5) If $i < N$ then
$i \leftarrow i + 1$, go to step 2
6) If $i = N$ and $l < L$ then
$i \leftarrow 1$, $l \leftarrow l + 1$, go to step 2
7) Compute physical measurements

The DQMC simulation consists of two stages: warmup stage and sampling stage. 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.

The Green's function associated with a configuration in Algorithm 1 is defined as

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

From this function, the Metropolis ratio $r_{l,i}$ can be easily computed thanks to the fact that $M_\sigma(h')$ is a rank-1 update of $M_\sigma(h)$. For example, at the spatial site $i = 1$:

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

and the Metropolis ratio r_{11} is given by

$$r_{11} = d_+ d_-,$$

where for $\sigma = \pm$,

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

and

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

Therefore, the gist of computing the Metropolis ratio r_{11} is to compute the $(1,1)$-element of the inverse of the matrix $M_\sigma(h)^{-1}$. If the Green's function $G^\sigma(h)$ has been computed explicitly in advance, then is essentially *free*, $O(1)$ operations, to compute the ratio r_{11}.

In the DQMC simulation, if the proposed h' is accepted, then the Green's function is updated by a rank-1 matrix:

$$G^\sigma(h) \leftarrow G^\sigma(h) - \frac{\alpha_{1,\sigma}}{r_{11}} u_\sigma w_\sigma^T$$

where

$$u_\sigma = (I - G^\sigma(h)) e_1$$

and

$$w_\sigma = (G^\sigma(h))^T e_1.$$

At the spatial site $i = 2$:

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

By a similar derivation as for the previous case, we have

$$r_{12} = d_+ d_-,$$

where for $\sigma = \pm$,

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

and

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

Correspondingly, if necessary, the Green's function is updated by the rank-1 matrix

$$G^\sigma(h) \leftarrow G^\sigma(h) - \frac{\alpha_{2,\sigma}}{r_{12}} u_\sigma w_\sigma^T$$

where

$$u_\sigma = (I - G^\sigma(h))e_2$$

and

$$w_\sigma = (G^\sigma(h))^T e_2.$$

In general, for $i = 3, 4, \ldots, N$, we can immediately see that the same procedure can be used for computing the Metropolis ratios r_{1i} and updating the Green's functions. In QUEST, this update of the Green's functions is delayed to lead to a block rank update instead of individual rank-1 updates [26]. After $i = N$, the Green's function cannot be updated anymore and it must recomputed from scratch. This computational kernel takes roughly 95% of the simulation execution time with previous DQMC implementations. The next two sections in this paper focus on numerical stability and performance improvements of the Green's functions evaluation in QUEST.

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. Following, we present general techniques implemented in QUEST to reduce its computational cost, exploiting the relation between successive evaluations.

A. Initial evaluation

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

$$G = (I + B_L B_{L-1} \cdots B_1)^{-1}, \tag{3}$$

where $B_i = B_{i,\sigma}(h_i)$ as defined in Eq. 2. Each B_i matrix is stored as

$$B_i = V_i B,$$

where $V_i = e^{\sigma\nu \mathrm{diag}(h_i)}$ 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 (3) is extremely ill-conditioned. Several methods have been proposed [27], [28], [19], [24] to stabilize the computation by stratifying the magnitude of elements in the matrix multiplications. All those methods inevitably require the pivoted QR decomposition. The stratification method proposed by Loh *et al* [19] is currently used in QUEST to calculate the Green's function G. In this algorithm, elements of different energy levels, which corresponds to different magnitude of numbers, are stratified by the pivoted QR decomposition.

Algorithm 2 Stratification method

1) Compute the pivoted QR: $B_1 = Q_1 R_1 P_1^T$
2) Set $U_1 = Q_1$, $D_1 = \mathrm{diag}(R_1)$ and $T_1 = D_1^{-1} R_1 P_1^T$
3) For $i = 2, 3, \cdots, L$
   a) Compute $C_i = (B_i U_{i-1}) D_{i-1}$
   b) Compute the pivoted QR: $C_i = Q_i R_i P_i^T$
   c) Set $U_i = Q_i$, $D_i = \mathrm{diag}(R_i)$, and $T_i = (D_i^{-1} R_i)(P_i^T T_{i-1})$
4) Compute $G = (T_L^{-T} Q_L^T D_b + D_s)^{-T} D_b Q_L^T$

At the last step D_b and D_i are computed from D_L as

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

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

The stability analysis of the stratification method can be found in [24]. This stratification process protects small numbers form being rounded 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 3 (a) instruct to first multiply B_i and U_{i-1}, and then this result by D_{i-1}. Using the typical parameters for DQMC simulations, B_i has not very big elements and U_{i-1} is a orthogonal matrix so their product could be computed accurately. As D_{i-1} is diagonal the second product is just a column scaling, without involving any addition, therefore can be also accurately computed using floating point arithmetic irrespective of the magnitude of the numbers. Obviously, we assume that these products can be represented without either overflow or underflow. In step 3 (c) the product $T_i = (D_i^{-1} R_i)(P_i^T T_{i-1})$ has also the same property. In this case the QR decomposition with pivoting make all $D_i^{-1} R_i$ elements about the same order of magnitude, therefore the whole product chain T_L can be accurately computed. The

splitting of D_L into D_b and D_s in the last step of the algorithm is also made for increasing the accuracy of the computation in the same way as before.

2) Matrix clustering: In order to reduce the computational cost of the Green's function evaluation, the stratification algorithm in QUEST works with $L_k = \lceil \frac{L}{k} \rceil$ matrices built by clustering k of the original B_i matrices. For example, if we define $\widehat{B}_1 = B_k B_{k-1} \cdots B_1$ we can transform the product

$$B_L B_{L-1} \cdots B_{k+1} B_k \cdots B_1 = B_L B_{L-1} \cdots B_{k+1} \widehat{B}_1.$$

Similarly, with $\widehat{B}_2 = B_{2k} B_{2k-1} \cdots B_{k+1}$ this product becomes

$$B_L B_{L-1} \cdots B_{2k} B_{2k-1} \cdots B_{k+1} \widehat{B}_1 = B_L B_{L-1} \cdots \widehat{B}_2 \widehat{B}_1.$$

Extending these definitions to all the sequence we obtain

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

where each $\widehat{B}_i$ is defined as

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

In this way the number of loop iterations in Algorithm 2 is reduced by a factor k. Usually, a value of $k \approx 10$ gives significant performance boost while maintaining numerical stability [29].

B. Green's function updating

1) Wrapping: As the simulation goes from one inverse temperature slice to the next, for example from $l = 1$ to $l = 2$ in Algorithm 1, the new Green's function takes the form

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

By this relationship, the new function can be cheaply computed from the previous G by noting that

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

This trick is called *wrapping* and can be use only for a few times $\ell \approx 10$ until the Green's function evaluation loses numerical accuracy, and must be re-computed by the stratification method (Algorithm 2) from scratch.

2) Computation recycling for matrix clustering: When using wrapping and matrix clustering with $k = \ell$ for the whole DQMC simulation (thousands of sweeps), the stratification algorithm computes a sequence of Green's functions such as

$$(I + \widehat{B}_{L_k} \widehat{B}_{L_k-1} \cdots \widehat{B}_1)^{-1}$$
$$(I + \widehat{B}_1 \widehat{B}_{L_k} \widehat{B}_{L_k-1} \cdots \widehat{B}_2)^{-1}$$
$$(I + \widehat{B}_2 \widehat{B}_1 \widehat{B}_{L_k} \widehat{B}_{L_k-1} \cdots \widehat{B}_3)^{-1}$$
$$\cdots$$
$$(I + \widehat{B}_{L_k-1} \widehat{B}_{L_k-2} \cdots \widehat{B}_1 \widehat{B}_{L_k})^{-1}$$

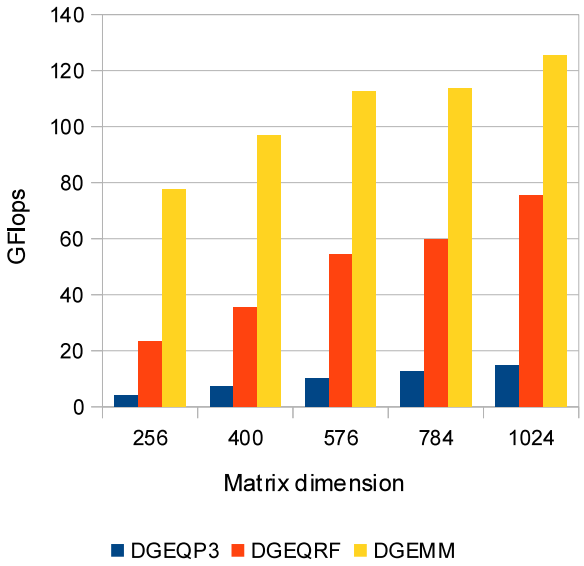

Figure 1. Performance comparison of relevant LAPACK routines using the Intel MKL library in a two socket 6 core Intel Westmere computer.

and starts over again. In all these cases, as the Metropolis algorithm only changes the rightmost matrix in the sequence, the rest of matrix clusters $\widehat{B}_i$ are the same on the next Green's function evaluation. In order to reduce the computational cost, QUEST stores these matrix clusters instead of recomputing them on each evaluation. Typical DQMC simulations requires storing less than one hundred matrices of size up to 1024×1024 (8 MBytes per matrix). This amount is not significant on current computers which easily have far more than 1 GByte of main memory.

IV. Multicore implementation

The main performance limit of the stratification algorithm is the extensive use of the QR decomposition with pivoting (line 3b of Algorithm 2). Current LAPACK implementations use BLAS level 3 operations for the trailing matrix update during the decomposition. However, the choice of pivots still requires a BLAS level 2 operation for each Householder reflector computed [25]. Therefore, the QR decomposition with pivoting (DGEQP3) is not a fully blocked algorithm like LU or regular QR decompositions (DGEQRF). This imposes a limit to the algorithm performance in current multicore architectures with deep memory hierarchies [30], as Figure 1 shows for the Intel MKL library running in a two socket 6 core Intel Westmere computer.

On this computer, matrix multiplication (DGEMM) obtains excellent performance even for small matrix sizes. Actually, DGEMM performance for 1024×1024 matrix is very close to the maximum GFlops rate for bigger matrices. DGEQRF performance is not as good as DGEMM

because of the impact of panel updates in the block QR decomposition for these matrix sizes [31], [32]. But even in this case, DGEQRF performance is much better than DGEQP3. Therefore, it would be interesting to modify the stratification algorithm so most calls to DGEQP3 could be replaced by DGEQRF, as the next section shows.

A. Stratification with pre-pivoting

As the stratification algorithm iterates the difference between largest and smallest elements in D_i increases, making the matrix C_i with an almost graded structure where the columns are ordered by its norm. In our experience, the QRP decomposition of C_i produces a permutation with few columns interchanges from the initial ordering.

Based on this observation, we propose a variant of the stratification algorithm where a regular QR is used instead of QRP and the permutation P is computed before the decomposition. This permutation is computed in the same way as the QRP algorithm, where matrix columns are sorted by decreasing norms. Our experimental results show that the QRP will interchange few columns during the decomposition and these permutations do not affect the result of the stratification procedure. This improved algorithm preserves the graded structure of C_i, but not as strong as in the original stratification. Also, the elements of $D_i^{-1}R_i$ are no longer close to one, but their magnitude is much smaller than the C_i matrix and not sufficient big to cause stability problems.

Algorithm 3 Stratification with prepivoting method

1) Compute the pivoted QR: $B_1 = Q_1 R_1 P_1^T$
2) Set $U_1 = Q_1$, $D_1 = \mathrm{diag}(R_1)$ and $T_1 = D_1^{-1} R_1 P_1^T$
3) For $i = 2, 3, \cdots, L$
   a) Compute $C_i = (B_i U_{i-1}) D_{i-1}$
   b) Compute the permutation P_i such as $\widehat{C}_i = C_i Pi$ has decreasing norm columns
   c) Compute a regular QR: $\widehat{C}_i = Q_i R_i$
   d) Set $U_i = Q_i$, $D_i = \mathrm{diag}(R_i)$, and $T_i = (D_i^{-1} R_i)(P_i^T T_{i-1})$
4) Compute $G = T_L^{-1}(U_L^T T_L^{-1} + D_L)^{-1} U_L^T$

This variant allows to exploit most of the processing power of current computers because most of the operations can now be expressed with blocked implementations: QR decompositions and matrix products.

B. OpenMP parallelization

Some operations in the previous algorithm are not available in the BLAS library or can be implemented more efficiently. The first kind of operations are the matrix scalings by rows or columns. These are present in two places (lines 3a and 3d) in Algorithm 3. Due to 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, assigning different columns to different processors. The second kind of operations are the computation of column norms for the pre-pivoting. Although BLAS provides an optimized routine for computing the norm of one column, there is not enough work in this operation to achieve good parallel performance. Our implementation uses OpenMP to compute several norms simultaneously and obtains better parallel efficiency.

These parallelizations do not achieve the full performance of all available cores because they are limited by memory bandwidth. However, as the matrices are relatively small with respect to cache sizes, our implementations can benefit from the memory hierarchy parallelism inside the multicore processors.

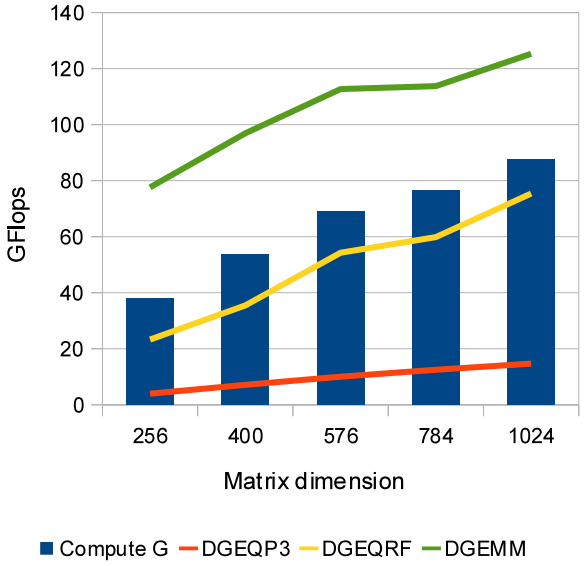

Figure 2. Green's function evaluation performance in GFlops.

C. Performance analysis

In this section, we show the performance of the improved Green's function evaluation in QUEST. Figure 2 shows the GFlops achieved for different matrix dimensions (number of sites N) and $L = 160$. These tests were run using a computer with two six core Intel Westmere processors at 2.93 GHz for a total of 12 cores. The performance reported is the average during a simulation that requires 1000 warming and 2000 measurements sweeps (Algorithm 1). These evaluations combine all the improvements presented so far in this paper, multicore implementation, stratification with pre-pivoting, wrapping and reuse of matrix clustering with $k = \ell = 10$.

The cost of Algorithm 3 is $O(N^3 L)$, however there are several operations like matrix scaling by a diagonal

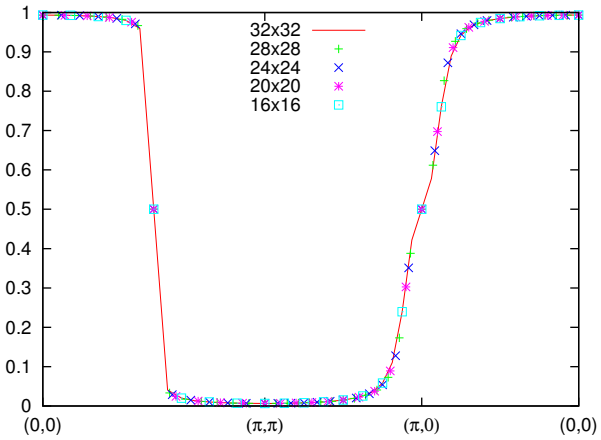

Figure 3. Mean momentum distribution $\langle n_{\mathbf{k}} \rangle$ of the two-dimensional Hubbard model at average density $\rho = 1$, interaction strength $U = 2$, and inverse temperature $\beta = 32$. $\langle n_{\mathbf{k}} \rangle$ is plotted along the momentum space symmetry line $(0,0) \rightarrow (\pi,\pi) \rightarrow (\pi,0) \rightarrow (0,0)$.

and norm computations (BLAS level 2 operations) with total cost $O(N^2 L)$. As N is relatively small, the impact of these operations is not negligible with respect of the $O(N^3 L)$ asymptotical cost. Taking into account this effect, the performance of QUEST Green's function evaluation is impressive, roughly 70% of the DGEMM GFlops rate and even better than the QR decomposition (DGEQRF).

V. Applications

To demonstrate the capability of the optimized QUEST package, we show in this section full simulation results for the two-dimensional Hubbard model. The performance of these full simulations will also be analyzed.

A. Hubbard model simulation results

We first focus on two important physical observables of the Hubbard model, namely, the momentum distribution

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

and the z-component spin-spin correlation function

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

In these equations, $n_{\mathbf{k},\sigma}$ and $n_{\mathbf{r},\sigma}$ are electron density operators in momentum and real space respectively, and $\sigma = \uparrow, \downarrow$.

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 material sciences. In Figure 3, 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 $(0,0) \rightarrow (\pi,\pi)$. 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 4 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.

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 5. However, in order to determine whether there is a true magnetic order in the bulk limit $N \rightarrow \infty$, 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 $N \rightarrow \infty$ limit to determine the existence of the magnetic structure in the bulk limit. While both panels in Figure 5 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. Performance analysis

Figure 6 shows the time required to compute 1000 warming and 2000 measurement steps for the previous simulations. The line in the plot is the nominal execution time predicted by the cost $O(N^3 L)$ of DQMC using the 256 sites simulation as a reference. The actual execution time for 256 sites is 1.25 hours but for 1024 sites is only 28 times as much (35.3 hours). This performance 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. However, this model does not take into account that linear algebra parallel efficiency improves as matrix size grows but still fits in cache memory. For the number of sites (matrix sizes) reported, this parallel performance increase is sufficiently big to partially compensate for the large asymptotical cost of DQMC simulations.

Table I shows the relative computation time for each of the different steps in the simulations presented before. The Green's function evaluation computational time is reduced from 95% of overall time in previous DQMC implementations to a 65% in QUEST, thanks to all the improvements presented in this paper.

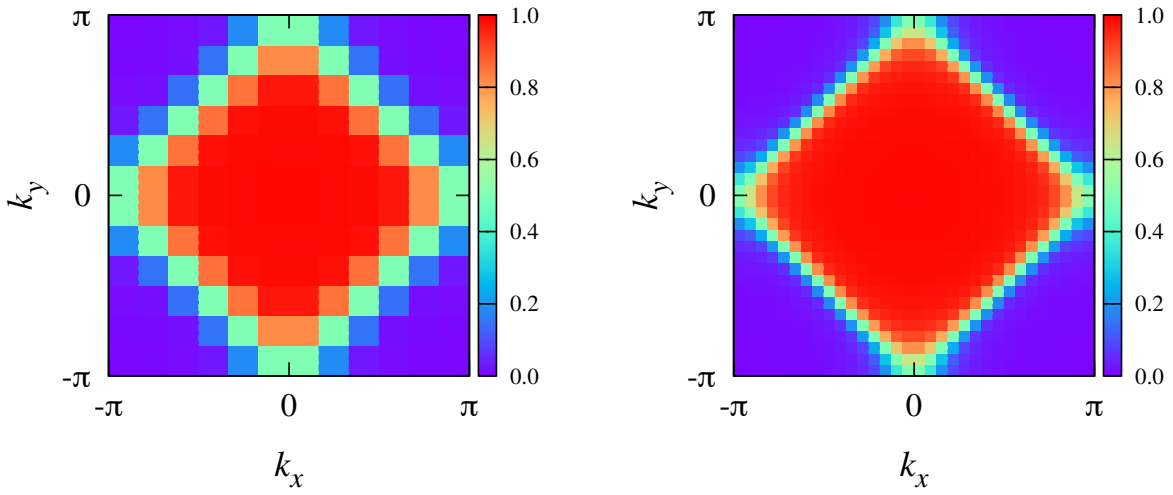

Figure 4. Color contour plot of the mean momentum distribution $\langle n_{\mathbf{k}} \rangle$ of the two-dimensional Hubbard model on 12×12 (left) and 32×32 (right) lattices. Simulation parameters are the same as Figure 3.

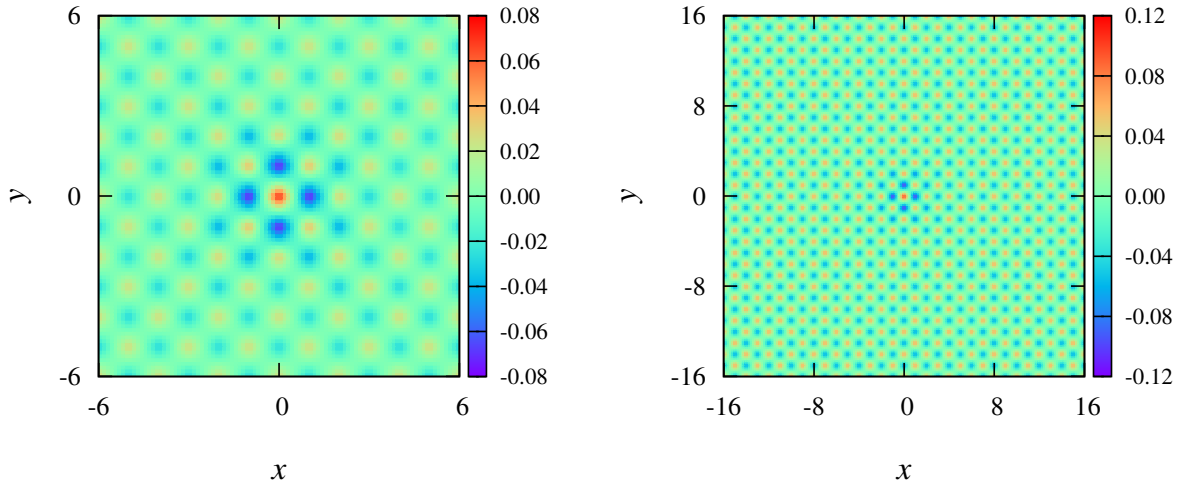

Figure 5. z-component spin-spin correlation $C_{zz}(\mathbf{r})$ on 12×12 (left) and 32×32 lattices with average density $\rho = 1$, interaction strength $U = 2$, and inverse temperature $\beta = 32$.

	Number of sites				
	256	400	576	784	1024
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.

VI. GPU ACCELERATION

Graphics processors currently offer more computational power than multicore processors. DQMC requires double precision computations for numerical stability but the previous GPUs offered single precision arithmetic. This has changed with the latest generation of the hardware, making GPUs a suitable option for DQMC simulations. Development tools and environments for GPUs are evolving towards greater simplicity of programming in each new hardware generation. However, GPUs are designed for a particular application and it still is more difficult to achieve good effi-

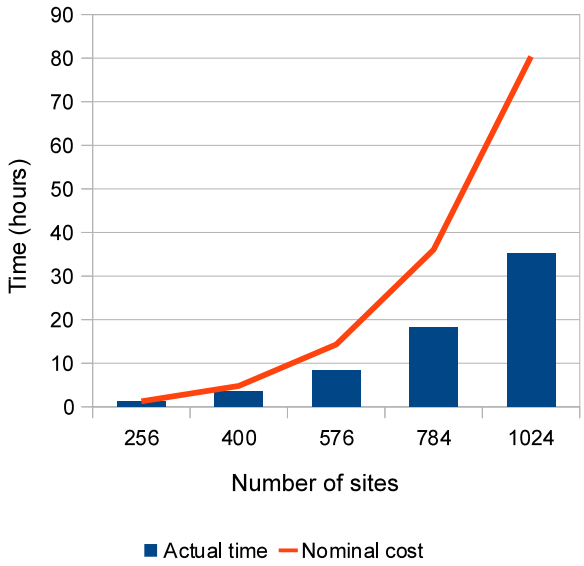

Figure 6. Actual time in QUEST and nominal (based on N^3 scaling) for a whole DQMC simulation with different number of sites.

ciency than with conventional multipurpose processors. The easiest way of using GPUs for numerical applications is by optimized libraries, such as NVIDIA CUBLAS. This library follows the classical BLAS concept, that is, it provides a standard interface for basic vector and matrix operations while hiding the actual details of implementation. This is very important for GPU because these details depend a lot on the hardware architecture, such as load balancing among processors and memory access patterns. In this section we show how to use a GPU to accelerate the clustering and wrapping of the Green's function evaluations discussed in sections III-B1 and III-A2.

A. GPU implementation details

Our implementation is based on the the latest DGEMM development for GPUs included in the CUBLAS [33], [34] library. However, this is not sufficient for achieving good performance in our application and we need to develop some customized CUDA code.

Algorithm 4 shows how to implement the product of several B_i matrices required by the matrix clustering. The main bottleneck when using a GPU for some parts of a computation is usually the data transfer between main and graphics memories. In DQMC simulations, the matrix B is fixed during all the process and it can be computed and stored at the start of the program. However, the diagonal matrices V_i could change and should be copied to the graphics memory each time. Also, the result A must be copied back to main memory. These transactions are relatively small,

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

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

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

```
i = blockIdx.x × blockDim.x + threadIdx.x
if i < n
    t = V_i
    for j = 1, 2, ..., n
        A_{i,j} = t × C_{i,j}
    end
end
```

$N \times L$ and $N \times N$ floating point values, and are not relevant compared to the whole algorithm execution time.

The computation of each $B_i = V_i B$ in Algorithm 4 is made by a copy of the B matrix and repeat calls to the vector scaling routine for each row. This trivial implementation is not quite efficient because these kind of BLAS level 1 routines are not able to achieve the full performance of the GPU. More important is that memory access pattern is row by row which does not exploit the coalescing memory access features of the graphics memory. Algorithm 5 is the CUDA kernel for computing efficiently this product $B_i = V_i B$.

With Algorithm 5, each thread of the GPU computes the scaling of one matrix row. This guarantees 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 in a GPU. Also, consecutive threads read and write in consecutive memory positions, technique known as coalesced access, which allows full memory bandwidth. Moreover, the number of memory reads is minimized by storing the V_i value in local memory for each thread. Like BLAS level 1 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 the sense of memory bandwidth. Last but not least, this procedure allows us to avoid the matrix copy in Algorithm 4.

Algorithm 6 Compute $A = B_i C B_i^{-1}$ using CUBLAS

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

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

```
i = blockIdx.x × blockDim.x + threadIdx.x
if i < n
    t = V_i
    for j = 1, 2, ..., n
        u = V_j (via texture)
        A_{i,j} = t × A_{i,j} / u
    end
end
```

The GPU can also be used for accelerating the wrapping step in DQMC simulations. Algorithm 6 is a implementation of $\widehat{G} = B_i G B_i^{-1}$ using the GPU via CUBLAS for all operations. As before, B and B^{-1} can be computed and stored in graphics memory at the start of the simulation. Also, 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 this scaling. This implementation is based on the previous 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.

B. Performance analysis

In this section, we show the impact of the GPU implementation proposed in the performance of Green's function evaluations for DQMC simulations. The tests were run using one node from the Carver cluster at NERSC using the Intel MKL and Nvidia CUBLAS libraries. These nodes have two Intel Nehalem processors at 2.4 GHz with four cores and a NVIDIA Tesla C2050 graphics processor with 448 computing elements.

Figure 7 compares the Gflops obtained by the GPU implementation we have presented, including memory transfer

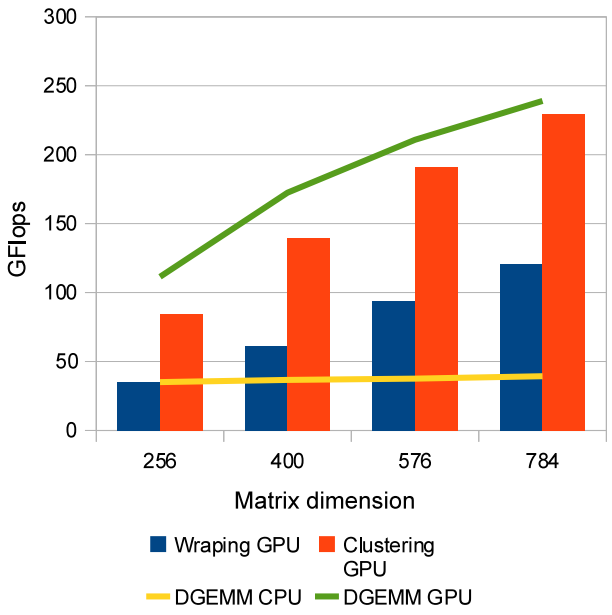

Figure 7. Gflops achieved for computing $\widehat{G} = B_i G B_i^{-1}$ (wrapping) and $\widehat{B} = B_{i+1} B_{i+2} \cdots B_{i+k}$ (blocking) compared to DGEMM CPU and GPU performance.

times, with the peak DGEMM CPU and GPU performance. This plot shows that our implementation for computing blocks $\widehat{B} = B_{i+1} B_{i+2} \cdots B_{i+k}$ is quite efficient, that is, close to GPU DGEMM performance, because several k matrix products are computed for each block with only one memory transfer. The GPU implementation for wrapping only computes two matrix products for each transfer and does not achieve this level of performance. However, its performance is always better than the CPU DGEMM, and its efficiency increases with the matrix dimension.

Figure 8 shows the performance of Green's function evaluations using the GPU for matrix clustering and wrapping, with different matrix dimensions (number of sites N) and $L = 160$. These evaluations combine all the improvements presented in this paper, multicore implementation, stratification with pre-pivoting, wrapping and reuse of matrix clustering with $k = \ell = 10$. The performance reported is the average for several runs during a simulation, and shows that the combination of a multicore processor with a GPU accelerator allows QUEST to obtain better performance than on a CPU only system.

VII. Conclusions

In this paper we present QUEST, a parallel implementation of the DQMC simulation on multicore processors. Parallelization of DQMC is extremely challenging due to the sequential nature of the underlying Markov chain and numerical stability issues. Although typical lattice sizes give

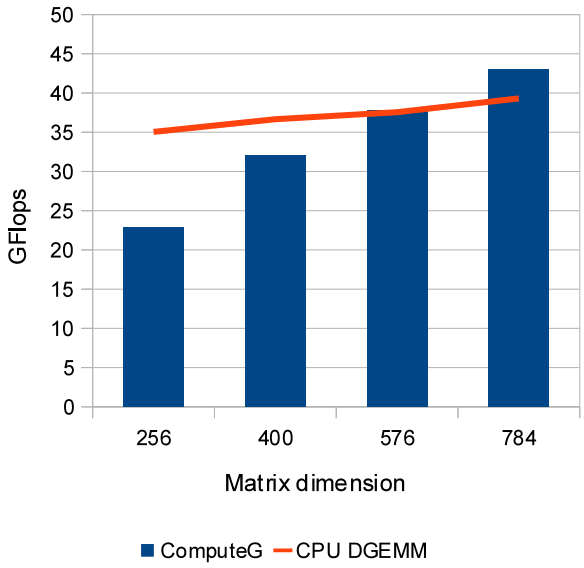

Figure 8. Green's function evaluation performance in GFlops per second.

matrices that are not sufficiently big to exploit large parallel computers, we show that current multisocket multicore processors can be quite efficient in this case.

The main kernel in DQMC simulation is the Green's function evaluation, which involves computing the inverse of a long chain of matrix multiplications. Traditional algorithms for these evaluations are based on graded decompositions generated by pivoted QR. In this paper we show that pivoted QR is far from efficient as the regular QR decomposition on current multicore processors. Therefore, we propose a novel algorithm which exploits the graded structure to spare the use of pivoting for the QR decomposition. The implementation of this algorithm using the MKL library and custom OpenMP code achieves an impressive 70% of DGEMM performance in a computer with two Intel Westmere 6 core processors.

We have integrated this new method and other improvements, such as some techniques to reuse computations for evaluating consecutive Green's functions, into QUEST. Using 36 hours on the same Intel processor, we are able to compute accurately the magnetic properties and Fermi surface of a system of $N = 1024$ electrons. This is almost an order of magnitude more difficult than current state of the art $N \approx 500$, owing to the N^3 scaling of the algorithms involved. The numerical stability of the QUEST implementation is validated by comparing simulations with previous results in the literature.

Furthermore, we show preliminary results using a GPU to accelerate parts of the DQMC simulation, showing promising performance. Our future research direction is to implement most of the DQMC simulation on the GPU using the recent work for QR decompositions on GPU [35], [36].

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.

VIII. Acknowledgments

This work was supported by the DOE SciDAC project under grant DOE-DE-FC0206ER25793 and National Science Foundation grant NSF PHY 1005502.

This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

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, Ed. New York: Plenum Press, 1995, NATO ASI series. Series B.

[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] *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 24,*, vol. 24, pp. 2278–2286, 1981.

[19] E. Loh and J. 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," LAPACK Working Note, Tech. Rep. 191, Sep. 2007.

[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 *Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, april 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. Philadelphia, PA: Society for Industrial and Applied Mathematics, 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*, 2010, doi:10.1016/j.laa.2010.06.023.

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

[26] M. Jarrell, private comunication.

[27] 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.

[28] 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.

[29] 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.

[30] Z. Drmač and K. Veselić, "New fast and accurate jacobi SVD algorithm: I," LAPACK Working Note, Tech. Rep. 169, Aug. 2005.

[31] B. Hadri, H. Ltaief, E. Agullo, and J. Dongarra, "Enhancing parallelism of tile QR factorization for multicore architectures." LAPACK Working Note, Tech. Rep. 222, Sep. 2009.

[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] R. Nath, S. Tomov, and J. Dongarra, "An improved Magma GEMM for Fermi graphics processing units," *International Journal of High Performance Computing Applications*, vol. 24, no. 4, pp. 511–515, 2010.

[34] J. Kurzak, S. Tomov, and J. Dongarra, "Autotuning GEMMs for Fermi," LAPACK Working Note, Tech. Rep. 245, Apr. 2011.

[35] 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 *Parallel Distributed Processing Symposium (IPDPS), 2011 IEEE International*, may 2011, pp. 932 –943.

[36] M. Anderson, G. Ballard, J. Demmel, and K. Keutzer, "Communication-avoiding QR decomposition for GPUs," *Parallel and Distributed Processing Symposium, International*, vol. 0, pp. 48–58, 2011.