

A Fast Selected Inversion Algorithm for Green's Function Calculation in Many-body Quantum Monte Carlo Simulations

Chengming Jiang
Dept. of Computer Science
University of California, Davis
Davis, CA USA
cmjiang@ucdavis.edu

Zhaojun Bai
Dept. of Computer Science
University of California, Davis
Davis, CA USA
bai@cs.ucdavis.edu

Richard Scalettar
Dept. of Physics
University of California, Davis
Davis, CA USA
scalettar@physics.ucdavis.edu

Abstract—The Hubbard Hamiltonian provides a theoretical framework for describing electron interactions of quantum many-body systems in condensed matter physics. Determinant Quantum Monte Carlo (DQMC) simulations of the Hubbard Hamiltonian have contributed greatly to understanding important properties of materials. Physical measurements such as superconductivity and magnetic susceptibility are based on selected entries of a large set of Green's functions. The computations of Green's functions are equivalent to computing selected blocks of the inverses of large-scale p-cyclic matrices. The performance of the state-of-art algorithm for computing Green's functions is around 100 Gflops on a 12-core Intel "Ivy Bridge" processor.

In this paper, we describe a fast selected inversion (FSI) algorithm for computing selected entries of Green's functions and present a parallel implementation using hybrid MPI/OpenMP programming. The FSI algorithm rests on three ideas: (1) applying a block cyclic reduction for a structure-preserving reduction; (2) computing the inverse of the reduced block p-cyclic matrix by a structured orthogonal factorization; (3) using the blocks of the inverse of the reduced block p-cyclic matrix as seeds to rapidly form the selected inversion in parallel. Performance results of the new FSI algorithm on Edison, NERSC (National Energy Research Scientific Computing Center)'s newest Cray XC30 supercomputer show a 80% improvement to 180 Gflops on the "Ivy Bridge" processor. The parallel applications of the FSI algorithm for computing selected entries of multiple Green's functions reach to 20–30 Tflops on 100 compute nodes with 2400 cores in total. The preliminary results show that the FSI algorithm speeds up a full DQMC simulation of Hubbard Hamiltonian of moderate size system by a factor of five, reducing from 2.5 hours down to less than half an hour on a compute node of our local system Zwolf with two 6-core Intel "Westmere" processors per node.

Keywords—p-cyclic matrix; Hubbard model; Quantum Monte Carlo simulations; Green's functions; Hybrid MPI/OpenMP parallelism

I. INTRODUCTION

Broadly speaking, theoretical and computational approaches to solve for the properties of condensed matter systems fall into two categories. Electronic structure methods attempt to solve the Schroedinger equation directly in continuum space. They can incorporate many of the details of specific materials, such as the precise chemical species

via the appropriate charges on the nuclei, but treat the interactions between the electrons through the rather crude Hartree-Fock approximation. Model Hamiltonians, on the other hand, consider the electrons as moving on discrete lattice sites, with typically only a very limited number of orbitals on each site. Indeed, in the most common case of the Hubbard Hamiltonian, a single orbital is considered. These models sacrifice chemical specificity, but allow for much more exact treatments of electron-electron interactions. Their most careful solution is via Quantum Monte Carlo, and is very time consuming. For example, some of the largest projects (hundreds of millions of core hours) of the DOE INCITE program¹ are QMC simulations of model Hamiltonians.

Determinant quantum Monte Carlo (DQMC) simulations [1], [2] have contributed greatly to the understanding of the Hubbard Hamiltonian. It provides one of the few means to attain exact solutions to the interacting electron problems which lie at the heart of phenomena like magnetism, superconductivity, metal-insulator and valence transitions in solids. In combination with density functional theory, DQMC methods are increasingly moving from providing qualitative insight concerning these dramatic phenomena to quantitative, material specific modeling. These approaches are very challenging, and their implementation constitutes one of the frontiers of modern computational science. Several recent advances [3] have already dramatically increased the number of electrons and material complexity that can be treated, but significant bottlenecks remain. Further algorithm development, and the implementation of these approaches on multi-core hardware, offer the prospect of breaking these logjams, and enabling the solution of frontier questions in the behavior of strongly correlated materials.

The state-of-art implementation of the DQMC simulation of the Hubbard model is available in the QUantum Electron Simulation Toolbox (QUEST)², a Fortran 90/95 package that uses two-dimensional periodic rectangular lattice as the

¹<http://www.doeleadershipcomputing.org/incite-program/>

²<https://code.google.com/p/quest-qmc/>

default geometry. The computational kernel in QUEST is the repeated computations of large number of Green's functions. Green's functions determine the probability amplitude for electrons to travel between sites, which is used for extracting information of phenomena caused by electron interaction such as magnetism, metal-insulator transitions and high-temperature superconductivity. In a MC simulation, a large number (order 10^3 to 10^4) of Green's functions are computed and used to calculate equal-time and time-dependent physical measurements.

In matrix computation terms, Green's function calculations concern computing selected blocks of the inverse of block p-cyclic matrices, which we refer to as Hubbard matrices. The Hubbard matrices are of dimension $NL \times NL$, where N is the number of spatial lattice sites and L is the number of time slices from the discretization of temporal domain which is proportional to the inverse temperature. To study moderate lattice at low temperature, $NL = O(10^3 \cdot 10^2) = O(10^5)$.

In QUEST, optimized BLAS and LAPACK subroutines have been used to execute matrix operations for computing Green's functions, which lead to certain performance improvement, but are not scalable on modern high-performance computers. In addition, physical measurement calculations are done by direct reference to elements of Green's functions in multi-layer loops and are bounded by communication cost. As a result, a modest size DQMC simulation with only 100 warmup iterations and 200 measurement iterations takes 2.5 hours on two 6-core "Westmere" processors, in which nearly 80% of the CPU time is spent on the computation of Green's functions and physical measurements.

Matrix inversion is one of the fundamental linear algebra problems. There is a large volume of literature on algorithms for computing selected entries of the inverse of a matrix, referred to as selected inversion. These algorithms can be organized in two classes. One concerns the selected inversion of unstructured sparse matrices [4], [5], [6], [7]. Another class concerns the selected inversion of structured matrices, such as Vandermonde [8], tridiagonal [9], [10], [11], [12] and Toeplitz [13], [14].

In this paper, we study an algorithm for computing selected blocks of the inverse of block p-cyclic matrices. Our contributions include (1) a fast selected inversion (FSI) algorithm to compute the selected blocks of the inverse of a block p-cyclic matrix by exploiting the structure of the matrix and underlying mathematical properties and (2) parallel application of the FSI algorithm on hybrid MPI/OpenMP by exploiting coarse-grain parallelism at the MPI level and fine-grain parallelism at the OpenMP level. Performance results of parallel applications of the FSI algorithm for computing selected entries of multiple Green's functions on Edison, NERSC (National Energy Research Scientific Computing Center)'s newest Cray XC30 supercomputer reach to 20–30 Tflops on 100 compute nodes with 2400 cores in total. The

preliminary results show that the FSI algorithm speeds up a full DQMC simulation of Hubbard Hamiltonian of moderate size system by a factor of five, reducing from 2.5 hours down to less than half an hour on the two 6-core Intel "Westmere" processors.

We note that there is a close relation between the FSI algorithm and the probing and sketching algorithms for matrix computations, such as the probing algorithm for computing the diagonal of the inverse of a sparse matrix inverse [15] and the trace of the inverse of a sparse matrix [16], [17], [18] and the matrix sketching methods for least squares regression and low rank approximation [19], [20].

II. FAST SELECTED INVERSION ALGORITHM

A. Green's function

In our current setting, Green's function can be defined by the inverse of the following block p-cyclic matrix in the normal form

$$A = \begin{bmatrix} A_{11} & & & A_{1L} \\ A_{12} & A_{22} & & \\ & \ddots & \ddots & \\ & & A_{L,L-1} & A_{LL} \end{bmatrix},$$

where each block is $N \times N$ square and the diagonal block matrices A_{ii} for $1 \leq i \leq L$ are nonsingular. The p-cyclic matrix has been studied since early 1950s [21]. It has been widely used in many applications such as numerical solution of partial differential equations [22], [23], Markov chain modelling [24] and QMC simulation.

Let $D = \text{diag}(A_{11}, A_{22}, \dots, A_{LL})$, then

$$M = D^{-1}A = \begin{bmatrix} I & & & B_1 \\ -B_2 & I & & \\ & \ddots & \ddots & \\ & & -B_L & I \end{bmatrix}, \quad (1)$$

where $B_1 = A_{11}^{-1}A_{1L}$ and $B_i = -A_{ii}^{-1}A_{i,i-1}$ for $2 \leq i \leq L$. A block LU factorization of M is given by $M = LU$ where

$$L = \begin{bmatrix} I & & & \\ -B_2 & I & & \\ & -B_3 & I & \\ & & \ddots & \ddots \\ & & & -B_L & I \end{bmatrix}$$

and

$$U = \begin{bmatrix} I & & & B_1 \\ & I & & B_2 B_1 \\ & & \ddots & \vdots \\ & & & I & B_{L-1} B_{L-2} \cdots B_1 \\ & & & & I + B_L B_{L-1} \cdots B_1 \end{bmatrix}.$$

It can be verified that the inverses of L and U are given by

$$L^{-1} = \begin{bmatrix} I & & & & \\ B_2 & & I & & \\ B_3 B_2 & & B_3 & & I \\ \vdots & & \vdots & \ddots & \vdots \\ B_L \cdots B_2 & B_L \cdots B_3 & \cdots & B_L & I \end{bmatrix}$$

and

$$U^{-1} = \begin{bmatrix} I & & & -B_1 F & \\ & I & & -B_2 B_1 F & \\ & & \ddots & \vdots & \\ & & & I & -B_{L-1} B_{L-2} \cdots B_1 F \\ & & & & F \end{bmatrix},$$

where $F = (I + B_L B_{L-1} \cdots B_2 B_1)^{-1}$. Consequently, the inverse of M , denoted by G , is then given by

$$G = M^{-1} = U^{-1} L^{-1} = (G_{k\ell}) \quad (2)$$

where for $1 \leq k, \ell \leq L$,

$$G_{k\ell} = W_{kk}^{-1} Z_{k\ell}, \quad (3)$$

and

$$W_{kk} = \begin{cases} I + B_k B_{k-1} \cdots B_1 B_L \cdots B_{k+1}, & 1 \leq k \leq L-1 \\ I + B_L B_{L-1} \cdots B_1, & k = L \end{cases}$$

and

$$Z_{k\ell} = \begin{cases} -B_k B_{k-1} \cdots B_1 B_L B_{L-1} \cdots B_{\ell+1}, & k < \ell < L \\ -B_k B_{k-1} \cdots B_1, & k < \ell = L \\ I, & k = \ell \\ B_k B_{k-1} \cdots B_{\ell+1}, & k > \ell \end{cases}.$$

The inverse of the block p-cyclic matrix A is then given by

$$A^{-1} = G D^{-1}.$$

Therefore, for the rest of discussion, we will focus on the computation of G , which is also the form of Green's function appeared in DQMC simulations.

A critical observation is that by the expression (3), there are relations between adjacent blocks of G . In other words, if a block $G_{k\ell}$ of G is known, then its adjacent blocks $G_{k-1,\ell}$, $G_{k+1,\ell}$, $G_{k,\ell-1}$ and $G_{k,\ell+1}$ can be easily computed. Specifically, if the block $G_{k\ell}$ is known, then vertically adjacent block $G_{k-1,\ell}$ satisfies the relation

$$G_{k-1,\ell} = B_k^{-1} G_{k\ell}, \quad (4)$$

except when $G_{k\ell}$ is on the boundaries,

- diagonal ($k = \ell \neq 1$): $G_{k-1,k} = B_k^{-1}(G_{kk} - I)$;
- first row ($k = 1, \ell \neq 1$): $G_{L\ell} = -B_1^{-1} G_{1\ell}$;
- corner ($k = 1, \ell = 1$): $G_{L1} = -B_1^{-1}(G_{11} - I)$.

Note that here and in the rest of paper, we use a torus index notation, namely if $k = 0$, then $k \equiv L$ and if $k = L + 1$, then $k \equiv 1$, which is the same for the index ℓ .

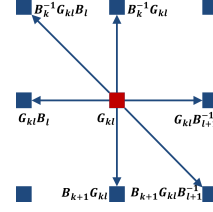


Figure 1. Relations between adjacent subblocks of Green's function.

Similarly,

$$G_{k+1,\ell} = B_{k+1} G_{k\ell}, \quad (5)$$

except when $G_{k\ell}$ is on the boundaries,

- sub-diagonal ($\ell = k+1$): $G_{k+1,k+1} = B_{k+1} G_{k,k+1} + I$;
- last row ($k = L, \ell \neq 1$): $G_{1\ell} = -B_1 G_{L\ell}$;
- corner ($k = L, \ell = 1$): $G_{11} = -B_1 G_{L1} + I$.

For horizontally adjacent block $G_{k,\ell-1}$, we have

$$G_{k,\ell-1} = G_{k\ell} B_\ell \quad (6)$$

except when $G_{k\ell}$ is on the boundaries,

- sub-diagonal ($\ell = k+1$): $G_{kk} = G_{k,k+1} B_{k+1} + I$;
- first column ($k \neq L, \ell = 1$): $G_{kL} = -G_{k1} B_1$;
- corner ($k = L, \ell = 1$): $G_{LL} = -G_{L1} B_1 + I$.

Similarly,

$$G_{k,\ell+1} = G_{k\ell} B_{\ell+1}^{-1} \quad (7)$$

except when $G_{k\ell}$ is on the boundaries,

- diagonal ($k = \ell \neq L$): $G_{k,k+1} = (G_{kk} - I) B_{k+1}^{-1}$;
- last column ($k \neq L, \ell = L$): $G_{k,1} = -G_{kL} B_1^{-1}$;
- corner ($k = L, \ell = L$): $G_{L1} = -(G_{LL} - I) B_1^{-1}$.

In addition, by the relations (4)–(7), the adjacent diagonal blocks $G_{k-1,\ell-1}$ and $G_{k+1,\ell+1}$ of $G_{k\ell}$ can be computed as

$$G_{k-1,\ell-1} = B_k^{-1} G_{k\ell} B_\ell \quad \text{and} \quad G_{k+1,\ell+1} = B_{k+1} G_{k\ell} B_{\ell+1}^{-1}.$$

The relations of adjacent blocks of Green's function are pictorially illustrated in Fig. 1

B. Selected inversion

A selected inversion is a collection of the selected blocks of G . Let us discuss four patterns of the selected inversion shown in Fig. 2. We use \mathcal{I} to denote the index set $\mathcal{I} = \{c-q, 2c-q, \dots, bc-q\}$, where c is a factor of L , $b = L/c$ and q is an integer randomly selected such that $0 \leq q \leq c-1$. q is chosen in the uniform distribution to allow blocks to be selected uniformly across the set $\{G\}$ of Green's functions.

- b diagonal blocks of G :

$$\mathcal{S}_1 = \{G_{kk} \mid k \in \mathcal{I}\}.$$

- b ($q \neq 0$) or $b-1$ sub-diagonal blocks of G :

$$\mathcal{S}_2 = \{G_{k,k+1} \mid k \in \mathcal{I} - \{L\}\}.$$

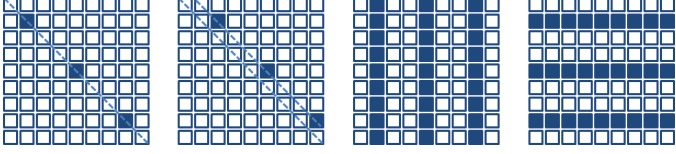


Figure 2. Patterns of selected inversions, diagonal, subdiagonal, columns and rows.

- b block columns of G :

$$\mathcal{S}_3 = \{G_{k\ell} \mid 1 \leq k \leq L \text{ and } \ell \in \mathcal{I}\}.$$

- b block rows of G :

$$\mathcal{S}_4 = \{G_{k\ell} \mid k \in \mathcal{I} \text{ and } 1 \leq \ell \leq L\}.$$

The set of selected inversion is relative small. For example, a selected inversion of column blocks only needs $1/c$ of the memory requirements of the the full inverse matrix. Typically for a p-cyclic matrix with $(N, L) = (1000, 100)$, we choose $c = \sqrt{L} = 10$. Thus we save the memory usage by 90%. A summary of the number of selected blocks in different patterns and the reduction factor compared with a full inversion is shown as follows:

Patterns	No. of selected blocks	Reduction factor
\mathcal{S}_1	b	cL
\mathcal{S}_2	b or $b - 1$	cL
\mathcal{S}_3	bL	c
\mathcal{S}_4	bL	c

C. Fast selected inversion algorithm

There are a number of algorithms aiming at computing selected diagonal blocks of Green's function. The method in [25] provides parallel approaches to compute the matrix chain multiplications arising in the explicit form of the diagonal blocks. The algorithm in [26] uses the pre-pivoting to balance the tradeoff between numerical stability and high-performance on multicore systems with GPU accelerations.

The computation of selected off-diagonal blocks of Green's function is much more challenging and has not been closely studied. In principle, one may use the explicit expression (3) to compute selected off-diagonal blocks of G . However, for example, it needs bL^2N^3 flops to compute the selected b block columns of G . In contrast, the fast selected inversion (FSI) algorithm described below reduces the flops by a factor of L , to $3bLN^3$.

The FSI algorithm rests on three ideas:

- applying the block cyclic reduction (BCR) for a structure-preserving reduction of M ;
- computing the inverse of the reduced block p-cyclic matrix by a stable structure orthogonal factorization;
- using adjacency relations (4)–(7) to rapidly form the selected inversion S .

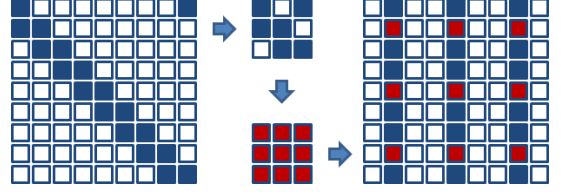


Figure 3. Graphical illustration of the FSI algorithm

The BCR is well-known, see for example [27]. In [28], Hirsch has exploited the BCR for computing the diagonal blocks (equal-time) of Green's functions.

At a high-level, the FSI algorithm is summarized as in Alg. 1, with a pictorial illustration in Fig. 3.

Algorithm 1 FSI algorithm

Input: M, c

Output: S

randomize $q \in \{0, \dots, c - 1\}$

$\widehat{M} = \text{CLS}(M, c, q)$

$\widehat{G} = \widehat{M}^{-1}$

$S = \text{WRP}(\widehat{G}, c, q)$

In Alg. 1, $\widehat{M} = \text{CLS}(M, c, q)$ is for a factor-of- c block-cyclic reduction of M , i.e.,

$$\widehat{M} = \begin{bmatrix} I & & & & \widehat{B}_1 \\ -\widehat{B}_2 & I & & & \\ & -\widehat{B}_3 & I & & \\ & & \ddots & \ddots & \\ & & & -\widehat{B}_b & I \end{bmatrix},$$

where \widehat{B}_i is a product of c consecutive matrices B_j , i.e.,

$$\widehat{B}_i = B_{j_0} B_{j_0+1} \cdots B_{j_0+c-1}.$$

where $j_0 = ci - q$. Note that if the index $j \leq 0$, then $j := j + L$.

The computational complexity of CLS is $2b(c-1)N^3$. Iterations for clustering \widehat{B}_i 's can be executed in embarrassingly parallel. We note that the integer c determines the size of clustering. A larger c leads to a greater reduction. However, the size of c is limited by numerical stability. A large c results in the loss of the precision due to round-off errors. Usually, $c \approx \sqrt{L}$. A numerical stability analysis for the choice of c can be found in [29].

The operation $\widehat{G} = \widehat{M}^{-1}$ in Alg. 1 is to use the block structured orthogonal factorization inversion (BSOFI) method from our early work [30] to compute the full inverse of the reduced block p-cyclic matrix \widehat{M} . It first applies the block structured orthogonal factorization $\widehat{M} = \widehat{Q}\widehat{R}$, and then calculate the inverse $\widehat{G} = \widehat{R}^{-1}\widehat{Q}^T$.

The BSOFI method is numerically stable and takes advantage of block p-cyclic structure of \widehat{M} to lower the

computational complexity to $7b^2N^3$. Instead of computing a full QR decomposition and then inversion of the p-cyclic matrix in the size of $(NL)^2$, the block structured orthogonal factorization computes the QR decomposition only on the dense blocks in the size of $2N \times N$ and then compute the inversion in the order of N . Thus, it exploits the sparsity and structure of the p-cyclic matrix.

The final step $\mathcal{S} = \text{WRP}(\hat{G}, c, q)$ in Alg. 1 is a *wrapping* process. By examining the explicit expression (3) for the blocks $G_{k\ell}$ of Green's function, the computed blocks of \hat{G} form a subset of the blocks of the original Green's function G , namely

$$\hat{G}_{k_0, \ell_0} = G_{ck_0-q, c\ell_0-q} \quad \text{for } 1 \leq k_0, \ell_0 \leq b. \quad (8)$$

This crucial observation leads us to use \hat{G}_{k_0, ℓ_0} as *seeds* to compute their adjacent blocks for forming the set \mathcal{S} of selected inversions of interest. Alg. 2 is a wrapping process for the selected block columns. The inner for loop is separated

Algorithm 2 Wrapping(WRP)

Input: \hat{G}, c, q

Output: \mathcal{S}

```

 $\mathcal{S} = \{\hat{G}_{k_0, \ell_0} | 1 \leq k_0, \ell_0 \leq b\}$ 
for each seed  $\hat{G}_{k_0, \ell_0}$  do
  set  $k = ck_0 - q$  and  $\ell = c\ell_0 - q$  in  $G$ 
  for  $i = 1, \dots, \lfloor (c-1)/2 \rfloor$  do
     $G_{k\ell} \rightsquigarrow G_{k-1, \ell}$  by Eq. 4
     $\mathcal{S} \leftarrow \mathcal{S} \cup \{G_{k-1, \ell}\}$ 
     $k = k - 1$ 
  end for
  reset  $k = ck_0 - q$  and  $\ell = c\ell_0 - q$  in  $G$ 
  for  $i = 1, \dots, \lfloor c/2 \rfloor$  do
     $G_{k\ell} \rightsquigarrow G_{k+1, \ell}$  by Eq. 5
     $\mathcal{S} \leftarrow \mathcal{S} \cup \{G_{k+1, \ell}\}$ 
     $k = k + 1$ 
  end for
end for

```

into two loops (for $G_{k\ell} \rightsquigarrow G_{k-1, \ell}$ and $G_{k\ell} \rightsquigarrow G_{k+1, \ell}$ respectively) to minimize the accumulated floating point arithmetic error. The computational cost is $3(bL - b^2)N^3$. Furthermore, we note that the b^2 iterations for calculating the adjacent blocks in wrapping are data independent. All seeds can be used independently to compute their adjacent blocks in parallel.

Computational costs depend on the shape of selected inversion. The computational complexity for explicit form 2 and FSI algorithm is shown as follows:

Selected inv.	Explicit form	FSI
b diagonals	$2b^2cN^3$	$[2(c-1) + 7b]bN^3$
$b-1$ sub-diag.	$4b^2cN^3$	$[2c + 7b]bN^3$
b cols./rows.	$b^3c^2N^3$	$3b^2cN^3$

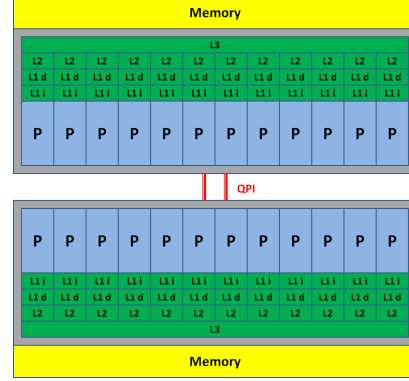


Figure 4. A Cray X30 dual-socket node, QuickPath Interconnect (QPI) connects two 12-core Intel “Ivy Bridge” processors.

Notes: (1) If we just compute the selected diagonals or sub-diagonal blocks, the major computation cost lies in BSOFI. (2) For most of the application, selected columns and rows are needed. In this case, the wrapping step is the bottleneck in term of the number of flops.

There are a number of advantages of the FSI algorithm. It uses less flops and reduces by a factor of $\frac{2}{3}bc^2$ and $\frac{7}{3}c$ than full LU inversion and BSOFI if b block columns are needed. More importantly, FSI can compute selected blocks of large scale p-cyclic matrices which may be not feasible by the full inversion method due to the memory bound. Comparing with directly applying the explicit form (3), say computing b columns, it is $\frac{1}{3}bc$ times faster. The main operations of the FSI algorithm are Level-3 BLAS operations, such as DGEMM. The process of FSI can be highly parallelized, which will be discussed in detail in §III.

III. HYBRID IMPLEMENTATION

A. OpenMP and MPI

Modern supercomputers have hierarchical architecture, where thousands of multi-socket multi-core shared-memory compute nodes are connected with a high-speed network. On each node, the memory hierarchy allows many cores to have multi-layer private cache and a big shared memory with non-uniform memory access. For example, NERSC's newest supercomputer Edison has 5576 compute nodes. With 24 cores per node, it has 133824 cores in total. An Edison compute node is shown in Fig. 4.

To take advantage of both distributed memory and multi-core shared memory architecture, it goes very natural to employ a hybrid MPI/OpenMP parallelism to a program that use MPI for message passing and use OpenMP for frequently shared data accessing. Generally, the hybrid model better matches the hierarchy memory structure, and many applications are found to be suitable for this hybrid model [31], [32], [33], [34]. However, there also exist some examples [31] where a pure MPI implementation is more efficient.

A tradeoff of hybrid model against pure MPI is that the extra communication overhead within each MPI process is replaced by OpenMP threads creation and synchronization.

As for all hybrid models, an important decision before launching the application is to select the number of OpenMP threads per MPI process and the number of MPI processes per node. Assigning too few MPI process with many OpenMP threads on a node may lead to poor performance [35], but assigning too many MPI process with few OpenMP threads on a node may exceed the memory capacity.

B. Parallel application of FSI algorithm

DQMC simulations require the selected inversion of tens of thousands of block p-cyclic matrices. The application of FSI algorithm is well positioned to fit the hybrid model. We can exploit two levels of parallelism. The first level on computing the inverse of multiple matrices is coarse-grained and is suitable for MPI. The second level on FSI itself is fine-grained, which is best suited for OpenMP. A complete pseudocode of parallel application of FSI algorithm is described in Alg 3.

Algorithm 3 Parallel application of FSI

Input: M_1, M_2, \dots, M_m and c
Output: S_1, S_2, \dots, S_m and global_measurement_quantities

```

On MPI_root {
  MPI_Init
  m_per_MPI = m/num_MPI_process
  MPI_Scatter (sbuff:{ $M_i$ },scount:m_per_MPI, ...)
}
On each MPI_process{
  for iter = 1, ..., m_per_task do
    select  $q \in \{0, \dots, c-1\}$  randomly
    !$omp parallel do
       $\hat{M} = \text{CLS}(M, c, q)$  by OpenMP multi-threads
    !$omp end parallel do nowait
     $\hat{G} = \hat{M}^{-1}$  by BSOFI
    initialize  $\mathcal{S} = \{\hat{G}_{k_0, \ell_0} | 1 \leq k_0, \ell_0 \leq b\}$ 
    !$omp parallel do
      execute WRP(Alg. 2) by OpenMP multi-threads
      compute local_measurement_quantities
    !$omp end parallel do nowait
  end for
  MPI_Reduce (sbuff:local_measurement_quantities,...)
}
On MPI_root{
  MPI_Finalize
  compute global_measurement_quantities
}

```

At the MPI level, a large set of p-cyclic matrices are distributed among the MPI processes. Each MPI process gets a portion ($m/\text{num_MPI_process}$) of the matrices and runs the FSI to collect the local measurement quantities.

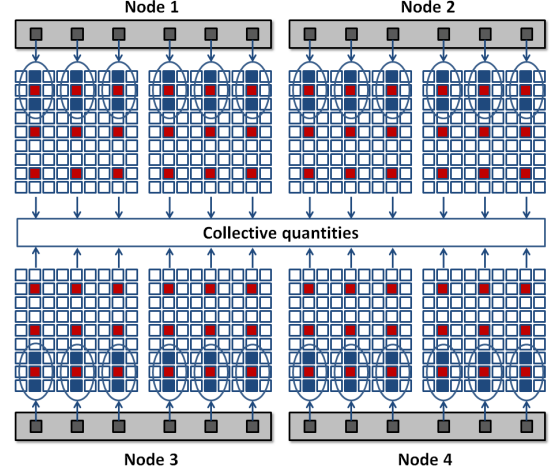


Figure 5. Hybrid MPI/OpenMP parallel application of the FSI algorithm for multiple Green's functions.

MPI_Reduce is called to collect the local measurement quantities to be aggregated into the global measurement quantities. Generating all the input matrices in one MPI process is neither efficient nor feasible due to the memory capacity when m is large. However, in the DQMC, the matrices are parameterized by an array of random parameters h , generated during a Monte Carlo process (see §IV). This allows us to generate a set of random parameters h on the MPI root process and scatter h to other MPI processes.

At the OpenMP level, that is the FSI algorithm. For clustering, the number of L B_i blocks are evenly divided into b clusters with c blocks each. Every OpenMP slave thread picks one cluster simultaneously to compute a chain of matrix multiplication in the size of c . For wrapping, \hat{G} are divided into b^2 seeds. Every OpenMP slave thread picks a seed individually and starts to calculate its adjacent blocks until the whole pattern of selected inversion is formed.

Note that the local measurement quantity calculations are carried out in the OpenMP parallel region. The reason to create local measurements for each thread is to overcome the concurrent writing issue caused by the data references of physical measurements, see an example in §IV.

Fig. 5 shows an example of topology of computing selected blocks of 8 matrices by 4 nodes. Each node has two MPI processes for 2 matrices respectively and each MPI process has 3 OpenMP threads associated with one matrix.

IV. QMC SIMULATION

At a high level, the DQMC simulation consists of two stages: warmup and physical measurement, as shown in Alg. 4. In the DQMC simulation, a DQMC sweep, as shown in Alg. 5, travels through every site of the lattice in a multi-layer imaginary time slices.

Algorithm 4 DQMC simulation

```

initialize HS configuration  $h_0 = (h_{\ell i}) = (\pm 1)$ 
%- Warmup stage -%
for  $i=1, \dots, w$  do
  DQMC sweep
end for
%- Measurement stage -%
for  $i=1, \dots, m$  do
  DQMC sweep
  compute Green's function and physical measurements
end for

```

Algorithm 5 DQMC sweep

```

for  $\ell = 1, 2, \dots, L$  do
  for  $i = 1, 2, \dots, N$  do
    (1) Propose a new configuration:  $h'_{\ell i} = -h_{\ell i}$ ;
    (2) Compute the Metropolis ratio:
      
$$r_{\ell i} = \frac{\det[M_+(h')] \det[M_-(h')]}{\det[M_+(h)] \det[M_-(h)]};$$

    (3) Apply Metropolis acceptance-rejection:
    randomize  $r \sim \text{uniform}[0, 1]$ ,
    if  $r \leq \min\{1, r_{\ell i}\}$  then
       $h = h'$ .
    end if
  end for
end for

```

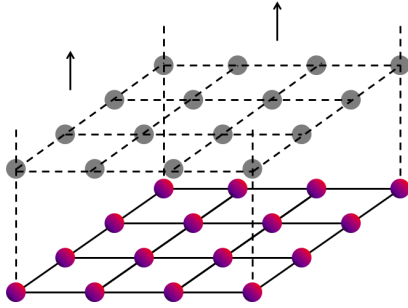


Figure 6. A $N = 4 \times 4$ lattice structure in a multi-layer imaginary time (L) slices

The physical measurements include the correlation functions for magnetic, charge, superconducting order and phase transitions and so on. They are classified by two categories. One is called equal-time measurements, which only need the data from the diagonal blocks of Green's functions G . The other one is called time-dependent measurements which need the information of off-diagonal blocks of G .

As an example, consider the measurement of XY spin-spin correlation (SPXX), an $L \times d_{\max}$ matrix with $d_{\max} \sim \mathcal{O}(N)$. To calculate the SPXX, we need to compute Green's functions for both spin direction at the same time [36], [37].

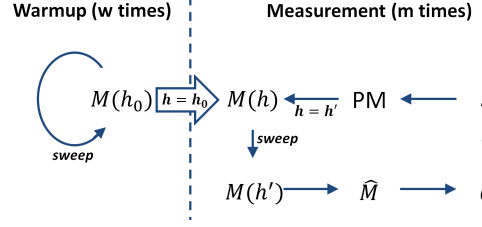


Figure 7. Overview of DQMC simulation

We denote G^σ with $\sigma = (\uparrow, \downarrow)$ to the electron spinning up and down, respectively. The (τ, d) element of SPXX contributed by G is given by

$$\{\text{SPXX}(G^\sigma)\}_{(\tau, d)} = -\frac{1}{2\mathcal{C}(\tau)} \sum_{(k, \ell)} \sum_{(i, j)} \left(G_{k\ell}^\uparrow(j, i) G_{\ell k}^\downarrow(i, j) + G_{k\ell}^\downarrow(j, i) G_{\ell k}^\uparrow(i, j) \right)$$

when $\mathcal{C}(\tau) > 0$, and equal to 0 when $\mathcal{C}(\tau) = 0$, where $\mathcal{C}(\tau)$ is the number of blocks contributing to $\{\text{SPXX}(G^\sigma)\}_{(\tau, d)}$,

$$\mathcal{C}(\tau) = \sum_{k=1}^N \sum_{\ell=1}^N b(k, \ell), \quad b(k, \ell) = \begin{cases} 1, & (k, \ell) \in T(\tau) \\ 0, & \text{otherwise} \end{cases}.$$

Index (k, ℓ) is in the set $T(\tau) = \{(k, \ell) | \mathcal{T}(k, \ell) = \tau\}$ where $\mathcal{T}(k, \ell)$ is a mapping from the block index (k, ℓ) to τ defined via temporal distances in lattices

$$\mathcal{T}(k, \ell) = \begin{cases} k - \ell, & k \geq \ell \\ k - \ell + L, & k < \ell \end{cases}.$$

Index (i, j) is in the set $D(d) = \{(i, j) | \mathcal{D}(i, j) = d\}$ where $\mathcal{D}(i, j)$ is a mapping from the entry index (i, j) to d defined via spatial distances in the lattice. Therefore, in order to compute $\{\text{SPXX}(G^\sigma)\}_{(\tau, d)}$, block columns and rows are both required (for entries in $G_{k\ell}$ and $G_{\ell k}$ simultaneously) from the selected inversion. We note that the calculations in $\{\text{SPXX}(G^\sigma)\}_{(\tau, d)}$ are element-wise. It is extremely inefficient level-1 BLAS operations. FSI enables these calculations be executed in OpenMP multi-threads.

By embedding the FSI algorithm for computing the Green's function and physical measurements, the overview of the DQMC simulation is shown in Fig. 7.

V. PERFORMANCE RESULTS

We begin with a validation of the correctness and accuracy of the FSI algorithm, and then report the performance of the FSI algorithm and its usage in full DQMC simulation.

Two platforms are used for performance testing. One is a Cray XC30 machine Edison at NERSC. Edison has 5576 compute nodes. With 24 cores per node, it has 133824 cores in total. Each Edison node consists of two sockets, and each socket is populated with a 12-core 2.4GHz Ivy Bridge processor. A node has 64GB DDR3 1866MHz memory (four 8GB DIMMS per socket). Each core has its own L1 and

L2 caches, with 64KB (32KB instruction cache and 32KB data cache) and 256KB respectively. A 30MB L3 cache is shared between 12 cores. Edison employs the “Dragonfly” topology for the interconnection network with 23.7TB/s global bandwidth. It has 0.25 μ s to 3.7 μ s MPI latency and 8 GB/sec MPI bandwidth.

The second platform is a single node of our local machine Zwolf. Each node has two 6-core 2.93GHz Intel Westmere Xeon X5670 processors. Each core has private 32KB L1 cache and 256KB L2 cache. Every six cores on a processor share a 12MB L3 cache. A node shares 96GB memory.

A. Correctness validation

To test the correctness of the FSI algorithm, we form a set of block p-cyclic Hubbard matrices M defined as (1). Each block B_ℓ is of the form

$$B_\ell(h(\ell, :)) = e^{t\Delta\tau K} e^{\sigma\nu V_\ell(h(\ell, :))},$$

where $h = (h_{\ell,i}) = (\pm 1)$ for $1 \leq \ell \leq L$ and $1 \leq i \leq N$ are random variables, referred to as a Hubbard-Stratonovich configuration in the DQMC simulation; t is a hopping amplitude; $\Delta\tau = \beta/L$, where β is the inverse temperature; $K = (k_{ij})$ is an adjacency matrix of the lattice structure; σ represents electron direction spinning; $\nu = \cosh^{-1} e^{\frac{U\Delta\tau}{2}}$, where U is the interacting energy; $V_\ell(h(\ell, :)) = \text{diag}(h(\ell, 1), h(\ell, 2), \dots, h(\ell, N))$.

We generate a random 6400 by 6400 p-cyclic Hubbard matrix $(N, L) = (100, 64)$ with $(t, \beta, \sigma, U) = (1, 1, 1, 2)$. The condition number of M is $O(10^5)$. We compute b selected block columns $\mathcal{S} = \{S_{ij}\}$ by FSI. G is computed by Intel MKL routines DGETRF and DGETRI. The correctness of the FSI algorithm is validated by the fact that the relative error

$$\varepsilon = \frac{1}{L \times b} \sum_{i=1}^L \sum_{j=1}^b \frac{\|S_{ij} - G_{i,cj-q}\|_F}{\|G_{i,cj-q}\|_F} = O(10^{-10}).$$

B. Performance of the FSI algorithm

We consider a set of Hubbard matrices with various block sizes $N = 256, 400, 576, 784, 1024$ and fixed $(L, c) = (100, 10)$. The set of the selected inversion is $b = L/c = 10$ block columns. The top plot of Fig. 8 shows the performance profile of three steps of OpenMP multi-threaded FSI on the “Ivy Bridge” processor. As we can see, the lower performance rate of the dense matrix inversions (BSOFI) is compensated by DGEMM-rich operations at the clustering and wrapping steps of FSI algorithm.

To test the scalability, we let $(N, L, c) = (576, 100, 10)$ and compute $b = L/c = 10$ block columns. The bottom plot of Fig. 8 shows the scalability of the FSI using OpenMP and MKL, respectively. We see that the former is much closer to the ideal scaling. The OpenMP overhead is negligible when the number of OpenMP threads per process is small.

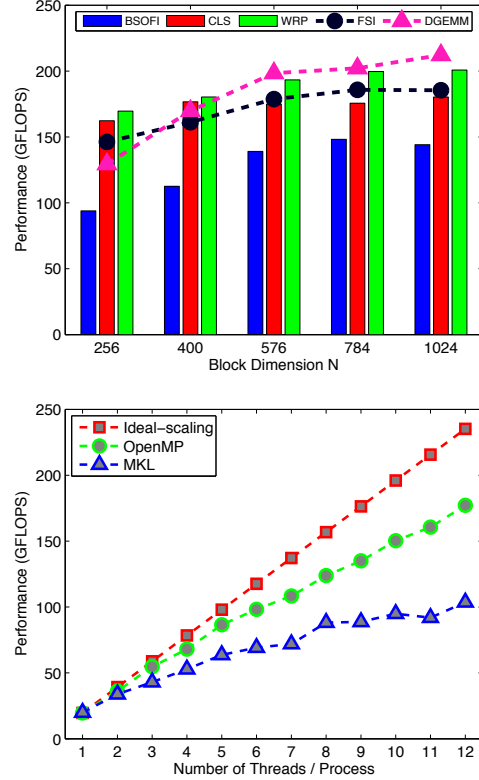


Figure 8. FSI performance rate (top) and scalability (bottom) on a single 12-core Intel “Ivy Bridge” processor.

For the performance test in hybrid MPI/OpenMP execution, we use 100 Edison nodes with a total of 2400 CPU cores to compute selected inversions of 2400 Hubbard matrices with $(L, c) = (100, 10)$ and different block sizes N . For each Hubbard matrix, $b = 10$ selected block columns are computed. Fig. 9 shows the performance rate with different MPI processes and OpenMP threads. We notice that each Edison compute node has a 32GB shared physical memory per socket (64GB in a node). Besides program itself, Node Linux kernel, Lustre file system software and message passing library buffers all consume memory. So available memory for one core is approximately 2.5GB. If an application runs too many MPI tasks on one node, it has a risk to exhaust the memory and an OOM (out of memory) killer will terminate the process on Edison.

By Fig. 9, we see that the pure MPI execution (with 1 OpenMP thread per MPI process) reaches the highest performance, but it is only applicable for block size $N = 400$. When $N = 576$, the memory requirement for the selected inversion is approximately 2.65GB. The execution of 12 MPI processes per socket requires 31.8GB that exceeds the available memory capacity on an Edison compute node. In this situation, the MPI and OpenMP hybrid model exploits the full usage of all available CPU cores and overcomes the

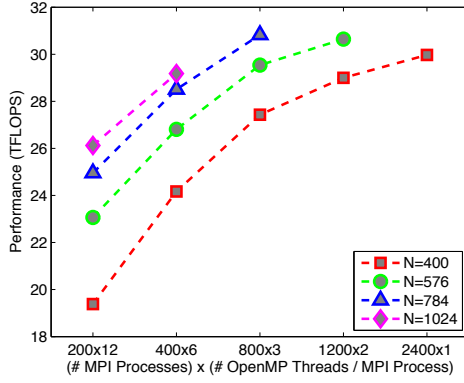


Figure 9. Performance rate of parallel application of FSI for multiple Green's functions with different numbers of MPI processes and OpenMP threads.

memory shortage to achieve the highest performance rate of 31 Tflops. The similar situation happens to $N = 784$ and $N = 1024$.

In summary, the performance of FSI algorithm with OpenMP is close to DGEMM-rate, a practical peak rate. In addition, FSI is scalable to the number of OpenMP threads and almost doubles the performance of pure multi-threaded MKL routines for computing the selected inversion. Moreover, when the parallel application of the FSI algorithm is used to computing the selected inversions of multiple Green's functions, the MPI/OpenMP model can maximize the power of thousands of available cores in the cases when the memory limits the number of MPI processes on each node.

C. FSI in DQMC

To examine the application of FSI in the DQMC simulation, we first integrate the FSI algorithm with the physical measurements. We consider a Hubbard matrix $M(h)$ of the dimensions $(L, N) = (100, 400)$. The cluster size $c = 10$. We compute all the diagonal blocks, b block rows and b block columns for both the equal-time and time-dependent measurements. We compare the CPU time of the serial execution and the parallel execution with OpenMP and pure MKL on a node of Zwolf. Fig. 10 shows a profile of CPU time of computing the Green's function and physical measurements. As we can see the pure MKL execution only reduces the CPU time for computing Green's function due to the power of multi-threaded optimized LAPACK routine. However, FSI with OpenMP is 90% less CPU time for the physical measurement calculations.

Finally, we test the impact of the FSI algorithm in a full DQMC simulation with $(N, L) = (400, 100)$. To limit the runtime, we set the number of warmup loops to $w = 100$ and the number of measurement loops to $m = 200$. The size of clustering in FSI is $c = 10$. Fig. 11 shows the total

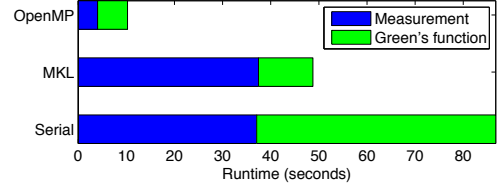


Figure 10. Runtime profile on a single Hubbard matrix

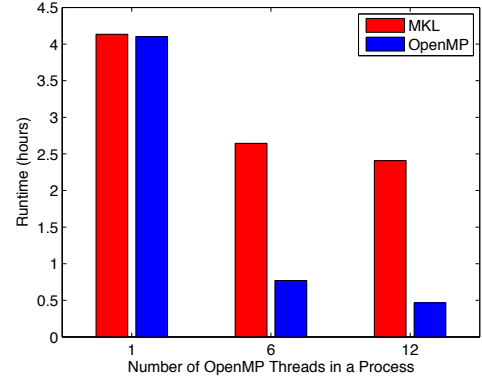


Figure 11. Runtime of a full DQMC simulation with $(w, m) = (100, 200)$

runtime of the DQMC with FSI on a node of Zwolf. As we can see, FSI with OpenMP gains a factor of 8 speedups from single-core to 12-core execution. In contrast, FSI with MKL only gains a factor of 1.6 speedups. As a result, the full DQMC simulation reduces from 2.5 hours to half an hour.

VI. CONCLUSION AND FUTURE WORK

In this paper, we tackled the bottleneck of Green's function calculations and physical measurements in many-body QMC simulations. The performance of FSI algorithm has doubled the performance of simple Intel MKL calls. Since the hybrid MPI/OpenMP model works perfectly on the parallel application of the FSI algorithm for multiple Green's function calculations and physical measurements, large scale DQMC simulations with production-graded physical measurements are feasible and importantly, more complex systems can be investigated. This is the work in progress. A promising by-product of this work is the extension of the basic idea of FSI algorithm to other types of structured matrices and applications.

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

REFERENCES

- [1] R. Blankenbecler, D. J. Scalapino, and R. L. Sugar, "Monte Carlo calculations of coupled boson-fermion systems. i," *Phys. Rev. D*, vol. 24, p. 2278, 1981.
- [2] R. R. d. Santos, "Introduction to quantum Monte Carlo simulations for fermionic systems," *Brazilian Journal of Physics*, vol. 33, pp. 36–54, 2003.
- [3] C.-C. Chang, S. Gogolenko, J. Perez, Z. Bai, and R. T. Scalettar, "Recent advances in determinant quantum monte carlo," *Philosophical Magazine*, vol. 95, pp. 1260–1281, 2015.
- [4] A. Erisman and W. Tinney, "On computing certain elements of the inverse of a sparse matrix," *Communications of the ACM*, vol. 18, pp. 177–179, 1975.
- [5] S. Li, S. Ahmed, G. Klimeck, and E. Darve, "Computing entries of the inverse of a sparse matrix using the FIND algorithm," *Journal of Computational Physics*, vol. 227, pp. 9408–9427, 2008.
- [6] D. E. Petersen, S. Li, K. Stokbro, H. H. B. Sørensen, P. C. Hansen, S. Skelboe, and E. Darve, "A hybrid method for the parallel computation of greena's functions," *Journal of Computational Physics*, vol. 228, pp. 5020–5039, 2009.
- [7] L. Lin, C. Yang, J. C. Meza, J. Lu, L. Ying, and E. Weinan, "Selin—an algorithm for selected inversion of a sparse symmetric matrix," *ACM Transactions on Mathematical Software*, vol. 37, p. 40, 2011.
- [8] M. E. El-Mikkawy, "Explicit inverse of a generalized Vandermonde matrix," *Applied Mathematics and Computation*, vol. 146, pp. 643–651, 2003.
- [9] R. K. Mallik, "The inverse of a tridiagonal matrix," *Linear Algebra and its Applications*, vol. 325, pp. 109–139, 2001.
- [10] G. Meurant, "A review on the inverse of symmetric tridiagonal and block tridiagonal matrices," *SIAM Journal on Matrix Analysis and Applications*, vol. 13, pp. 707–728, 1992.
- [11] P. Schlegel, "The explicit inverse of a tridiagonal matrix," *Mathematics of Computation*, vol. 24, p. 665, 1970.
- [12] S. Vatsya and H. Pritchard, "An explicit inverse of a tridiagonal matrix," *International journal of computer mathematics*, vol. 14, pp. 295–304, 1983.
- [13] C. Da Fonseca and J. Petronilho, "Explicit inverse of a tridiagonal k -Toeplitz matrix," *Numerische Mathematik*, vol. 100, pp. 457–482, 2005.
- [14] W. F. Trench, "An algorithm for the inversion of finite Toeplitz matrices," *Journal of the Society for Industrial & Applied Mathematics*, vol. 12, pp. 515–522, 1964.
- [15] J. M. Tang and Y. Saad, "A probing method for computing the diagonal of a matrix inverse," *Numerical Linear Algebra with Applications*, vol. 19, pp. 485–501, 2012.
- [16] Z. Bai, M. Fahey, and G. Golub, "Some large-scale matrix computation problems," *Journal of Computational and Applied Mathematics*, vol. 74, pp. 71–89, 1996.
- [17] H. Avron and S. Toledo, "Randomized algorithms for estimating the trace of an implicit symmetric positive semi-definite matrix," *Journal of the ACM*, vol. 58, p. 8, 2011.
- [18] A. Stathopoulos, J. Laeuchli, and K. Orginos, "Hierarchical probing for estimating the trace of the matrix inverse on toroidal lattices," *SIAM Journal on Scientific Computing*, vol. 35, pp. S299–S322, 2013.
- [19] E. Liberty, "Simple and deterministic matrix sketching," in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2013, pp. 581–588.
- [20] D. P. Woodruff, "Sketching as a tool for numerical linear algebra," *arXiv preprint arXiv:1411.4357*, 2014.
- [21] R. S. Varga, " p -cyclic matrices: A generalization of the Young-Frankel successive overrelaxation scheme," *Pacific Journal of Mathematics*, vol. 9, pp. 617–628, 1959.
- [22] O. G. Ernst, "Equivalent iterative methods for p -cyclic matrices," *Numerical Algorithms*, vol. 25, pp. 161–180, 2000.
- [23] G. Fairweather and I. Gladwell, "Algorithms for almost block diagonal linear systems," *SIAM Rev.*, vol. 46, pp. 49–58, 2004.
- [24] W. J. Stewart, *Introduction to the numerical solution of Markov chains*. Princeton, NJ: Princeton Univ. Press, 1994, vol. 41.
- [25] C.-R. Lee, I.-H. Chung, Z. Bai *et al.*, "Parallelization of dqmc simulation for strongly correlated electron systems," in *Parallel & Distributed Processing (IPDPS)*, 2010 IEEE International Symposium on, 2010, pp. 1–9.
- [26] A. Tomas, C.-C. Chang, R. Scalettar, and Z. Bai, "Advancing large scale many-body QMC simulations on GPU accelerated multicore systems," in *Parallel & Distributed Processing Symposium (IPDPS)*, 2012 IEEE 26th International. IEEE, 2012, pp. 308–319.
- [27] W. Gander and G. H. Golub, "Cyclic reduction – history and applications," *Scientific Computing*, pp. 73–85, 1997.
- [28] J. E. Hirsch, "Stable Monte Carlo algorithm for fermion lattice systems at low temperatures," *Phys. Rev. B*, vol. 38, p. 12023, 1988.
- [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*, ser. Contemporary Applied Mathematics, T. Y. Hou, C. Liu, and J.-G. Liu, Eds. Beijing: Higher Education Press and World Scientific, 2009, vol. 12, ch. 1, pp. 1–100.
- [30] S. Gogolenko, Z. Bai, and R. Scalettar, "Structured orthogonal inversion of block p -cyclic matrices on multicores with GPU accelerators," in *Euro-Par 2014 Parallel Processing*. Switzerland: Springer, 2014, pp. 524–535.
- [31] J. Diaz, C. Munoz-Caro, and A. Nino, "A survey of parallel programming models and tools in the multi and many-core era," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 23, pp. 1369–1386, 2012.
- [32] M. J. Chorley and D. W. Walker, "Performance analysis of a hybrid MPI/OpenMP application on multi-core clusters," *Journal of Computational Science*, vol. 1, pp. 168–174, 2010.
- [33] H. Jin, D. Jespersen, P. Mehrotra, R. Biswas, L. Huang, and B. Chapman, "High performance computing using MPI and OpenMP on multi-core parallel systems," *Parallel Computing*, vol. 37, pp. 562–575, 2011.
- [34] G. Schubert, G. Hager, H. Fehske, and G. Wellein, "Parallel sparse matrix-vector multiplication as a test case for hybrid MPI + OpenMP programming," in *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW)*, 2011 IEEE International Symposium on, 2011, pp. 1751–1758.
- [35] G. Hager and G. Wellein, *Introduction to high performance computing for scientists and engineers*. Boca Raton, FL: CRC Press, Taylor and Francis Group, 2010.
- [36] N. W. Ashcroft and N. D. Mermin, *Solid State Physics*. Fort Worth: Cengage Learning, 1976.
- [37] M. Tinkham, *Group theory and quantum mechanics*. Courier Dover Publications, 2003.