2016 IEEE International Parallel and Distributed Processing Symposium

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 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 block entries 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, National Energy Research Scientific Computing Center (NERSC)'s Cray XC30 supercomputer, show an 80% improvement to 180 Gflops on the Intel "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. The preliminary results show that the FSI algorithm speeds up a full DQMC simulation of the Hubbard Hamiltonian by a factor of five, reducing from three and a half hours down to only forty minutes on the 12-core processor.

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

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 nucleii, 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. These models allow for much more exact treatments of electron-electron interactions. Model Hamiltonians can be solved by approximate methods like mean field theory, which often get the qualitative physics correct, but are wrong quantitatively. They can also be solved exactly on very small clusters of $N \approx 10$ sites by explicitly enumerating all the states of the quantum system, and diagonalizing a matrix whose dimension grows exponentially with N. Quantum Monte Carlo (QMC) allows, in many important cases, an exact solution but on lattices an order of magnitude larger, $N \approx 10^3$ sites. More precisely, QMC gives exact results for observables within statistical error bars which can be made systematically smaller by increasing the number of samples generated. However, QMC is very time consuming. For example, some of the largest projects (hundreds of millions of core hours) of the DOE INCITE program are QMC simulations of model Hamiltonians.

Our specific method, Determinant QMC (DQMC) [1], [2] works on real space lattices of finite size. An alternate approach, the "dynamic cluster approximation" (DCA) also solves interacting electron model Hamiltonians, but works instead on discrete grids in momentum space. The two approaches provide useful complementary information. DQMC provides better representation of spatial correlation functions. The DCA has better performance at low temperatures, and often provides more simple routes to locating phase transitions. A lot of important work has been done on porting the DCA to high performance computing platforms resulting in some of the most accurate information on the physics of correlated electron systems currently available [3]. But there are few studies of high-performance DQMC.

DQMC methods are increasingly moving from providing qualitative insight concerning the dramatic phenomena like magnetism, superconductivity, metal-insulator and valence transitions in solids, to quantitative, material specific modeling. These approaches are very challenging, and their implementation constitutes one of the frontiers of modern computational science. Some recent advances [4] 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)[1], a Fortran 90/95 package that uses two-dimensional periodic rectangular lattice as the 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, $NL \approx 10^3 \cdot 10^2$.

In QUEST, optimized BLAS and LAPACK have been used to execute matrix operations for computing Green's functions, which lead to performance improvement, but are not scalable. In addition, physical measurements 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 warmups and 200 measurements takes three and a half hours on a 12-core Intel "Ivy Bridge" processors, in which nearly 80% of the CPU time is spent on the computation of Green's functions and physical measurements.[2]

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 unstructured sparse matrices [5], [6], [7]. Another concerns the selected inversion of structured matrices, such as Vandermonde [8], tridiagonal [9], [10] and Toeplitz [11], [12].

[1]https://code.google.com/p/quest-qmc/

[2]The length of a DQMC simulation depends on the accuracy desired for the measurements of interest and also on parameters like the interaction strength and temperature. With that caveat, simulations of length 1000 warmups and 2000 measurements are roughly typical.

In this paper, we study an algorithm for computing selected blocks of the inverse of block p-cyclic matrices. Our contributions include (a) 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 (b) a hybrid MPI/OpenMP implementation of the FSI algorithm through exploiting coarse-grain parallelism at the MPI level and fine-grain parallelism at the OpenMP level. Performance results of the parallel FSI algorithm for computing selected entries of multiple Green's functions on Edison, NERSC's Cray XC30 supercomputer reach to 20–30 Tflops on 100 compute nodes with 2400 cores. 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 three and a half hours down to only forty minutes on a 12-core Intel "Ivy Bridge" processor.

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 [13] and the trace of the inverse of a sparse matrix [14], [15], [16] and the matrix sketching methods for least squares regression and low rank approximation [17].

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

$$A = \begin{bmatrix} A_{11} & & & A_{1L} \\ A_{21} & A_{22} & & \\ & \ddots & \ddots & \\ & & A_{L,L-1} & A_{LL} \end{bmatrix},$$

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 [18]. It has been widely used in many applications such as numerical solution of partial differential equations [19], [20], Markov chain modelling [21] and QMC simulation [1], [2].

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

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

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

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

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

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

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

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

and

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

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

$$A^{-1} = GD^{-1}.$$

Therefore, for the rest of discussion, we will focus on the computation of G, which is also the form of Green's function that 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 upper block $G_{k-1,\ell}$ satisfies the relation

$$G_{k-1,\ell} = B_k^{-1} G_{k\ell}, \tag{4}$$

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.

Similarly, vertically adjacent lower block $G_{k+1,\ell}$ of $G_{k,\ell}$ satisfies

$$G_{k+1,\ell} = B_{k+1} G_{k\ell}, \tag{5}$$

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 left block $G_{k,\ell-1}$ of $G_{k,\ell}$, we have

$$G_{k,\ell-1} = G_{k\ell} B_\ell \tag{6}$$

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, horizontally adjacent right block $G_{k,\ell+1}$ of $G_{k,\ell}$ is given by

$$G_{k,\ell+1} = G_{k\ell} B_{\ell+1}^{-1} \tag{7}$$

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 ($k = L, \ell = L$): $G_{L1} = -(G_{LL} - I) B_1^{-1}$.

Furthermore, 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

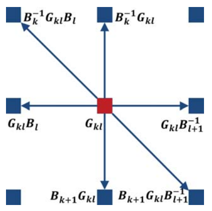


Figure 1. Relations between adjacent subblocks of Green's function.

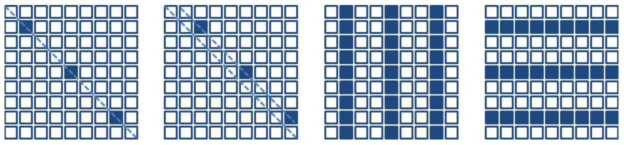


Figure 2. Patterns of selected inversions, diagonal, subdiagonal, columns and rows.

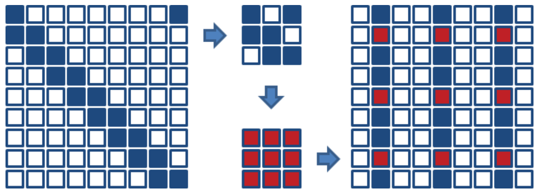


Figure 3. Graphical illustration of the three stages of the FSI algorithm.

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 $\mathcal{I} = \{c-q,\, 2c-q,\, \ldots,\, bc-q\}$, 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 uniformally across a set of Green's functions.

- b diagonal blocks of G:
$$\mathcal{S}_1 = \{G_{kk} \,|\, k \in \mathcal{I}\}.$$
- b $(q \neq 0)$ or $b-1$ sub-diagonal blocks of G:
$$\mathcal{S}_2 = \{G_{k,k+1} \,|\, k \in \mathcal{I} - \{L\}\}.$$
- b block columns of G:
$$\mathcal{S}_3 = \{G_{k\ell} \,|\, 1 \leq k \leq L \text{ and } \ell \in \mathcal{I}\}.$$
- b block rows of G:
$$\mathcal{S}_4 = \{G_{k\ell} \,|\, k \in \mathcal{I} \text{ and } 1 \leq \ell \leq L\}.$$

In applications, the set of selected blocks is relatively small. For example, a selected inversion of column blocks only needs $1/c$ of the memory for storing 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 [22] provides parallel approaches to compute the matrix chain mutiplications arising in the explicit form (3) of diagonal blocks. The algorithm in [23] 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 the following three ideas:

- applying the block cyclic reduction (BCR) for a structure-preserving reduction of the Hubbard matrix 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 $\mathcal{S}$.

The BCR is well-known, see for example [24]. In [25], Hirsch has exploited the BCR for computing the diagonal blocks (equal-time) of Green's functions, although it was not explicitly stated.

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: $\mathcal{S}$
  randomize $q \in \{0, ..., c-1\}$
  $\widehat{M} = \text{CLS}(M, c, q)$
  $\widehat{G} = \widehat{M}^{-1}$
  $\mathcal{S} = \text{WRP}(\widehat{G}, c, q)$
```

In Alg. 1, $\widehat{M} = \text{CLS}(M, c, q)$ is for a factor-of-c BCR 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 [26].

The operation $\widehat{G} = \widehat{M}^{-1}$ in Alg. 1 is to compute the full inverse of the reduced block p-cyclic matrix $\widehat{M}$ by using a block structured orthogonal factorization inversion (BSOFI) method from our early work [27]. The BSOFI method 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 fully exploits the structure of the p-cyclic matrix.

The final step $\mathcal{S} = \text{WRP}(\widehat{G}, c, q)$ 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 $\widehat{G}$ form a subset of the blocks of the original Green's function G, namely

$$\widehat{G}_{k_0,\ell_0} = G_{ck_0-q,c\ell_0-q} \quad \text{for } 1 \le k_0, \ell_0 \le b. \tag{8}$$

This crucial observation leads us to use $\widehat{G}_{k_0,\ell_0}$ 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 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.

The computational cost of the FSI algorithm depends on the shape of selected inversion. In the following table, we compare the computational complexity the explicit inversion using the expression (2) and the FSI algorithm for the four patterns of the selected inversions discussed in Sec.II-B:

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.	$b^3c^2N^3$	$3b^2cN^3$

Notes: (a) If we just compute the selected diagonals or sub-diagonal blocks, the major computation cost lies in BSOFI. (b) For most of the applications, selected columns and rows

Algorithm 2 Wrapping(WRP)

```
Input: $\widehat{G}, c, q$
Output: $\mathcal{S}$
  $\mathcal{S} = \{\widehat{G}_{k_0,\ell_0} | 1 \le k_0, \ell_0 \le b\}$
  for each seed $\widehat{G}_{k_0,\ell_0}$ do
    set $k = ck_0 - q$ and $\ell = c\ell_0 - q$ in $G$
    for $i = 1, ..., \lfloor (c-1)/2 \rfloor$ 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 $i = 1, ..., \lfloor c/2 \rfloor$ 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
```

are needed. In this case, the wrapping step is the bottleneck in terms 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 expression (3), say computing b columns, it is $\frac{1}{3}bc$ times faster. The main operations of the FSI algoirthm are Level-3 BLAS operations, such as `DGEMM`. The FSI algorithm can be highly parallelized, which will be discussed in detail in Sec.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 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 naturally to employ hybrid MPI/OpenMP parallelism that uses MPI for message passing and OpenMP for frequently shared data accessing. Many applications are found to be suitable for the hybrid model [28], [29], [30].However, there also exist some examples where a pure MPI implementation is more efficient [28]. A tradeoff of hybrid model against pure MPI is that the

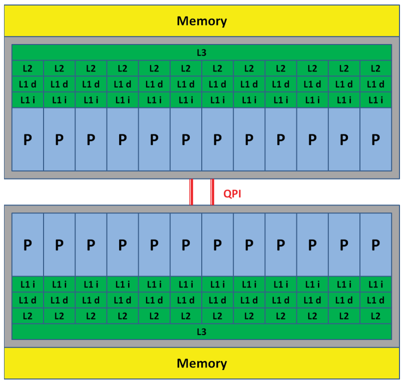


Figure 4. A Cray X30 dual-socket node, QuickPath Interconnect (QPI) connects two 12-core Intel "Ivy Bridge" processors.

extra communication overhead within each MPI process is replaced by OpenMP threads creation and synchronization. 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 processes with many OpenMP threads on a node may lead to poor performance, but assigning too many MPI processes with few OpenMP threads on a node may exceed the memory capacity [31].

B. *Parallel implementation of FSI algorithm*

DQMC simulations require the selected inversions of tens of thousands of block p-cyclic matrices. The implementation of the FSI algorithm is well positioned to fit the hybrid model. We can exploit two levels of parallelism. The first level on computing the inverses 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 implementation of the FSI algorithm for a set of matrices is described in Alg. 3.

At the MPI level, a large set of p-cyclic matrices are distributed among the MPI processes. Each MPI process gets a portion (m/num_MPI_process) of the matrices and runs the FSI to collect the local measurement quantities. `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. Fortunately, in the DQMC, the matrices are parameterized by an array of random parameters h, generated during a Monte Carlo process (see Sec.IV). This allows us to generate a set of random parameters h on the MPI root process and scatter h to other MPI processes. The FSI algorithm on a single matrix M is implemented by OpenMP. At the clustering step of FSI, the number L of B_i blocks are evenly divided into b clusters with c blocks each. Every OpenMP slave thread picks one cluster simultaneously

Algorithm 3 Parallel application of FSI

```
Input: $M_1, M_2, ..., M_m$ and $c$
Output: $\mathcal{S}_1, \mathcal{S}_2, ..., \mathcal{S}_m$ 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_MPI do
    select $q \in \{0, ..., c-1\}$ randomly
    !$omp parallel do
      $\widehat{M} =$ CLS$(M, c, q)$ by OpenMP multi-threads
    !$omp end parallel do nowait
    $\widehat{G} = \widehat{M}^{-1}$ by BSOFI
    initialize $\mathcal{S} = \{\widehat{G}_{k_0,\ell_0} | 1 \leq k_0, \ell_0 \leq b\}$
    !$omp parallel do
      execute WRP(Alg. 2) by OpenMP multi-threads
      compute local_measurement_quantities
    !$omp end parallel do nowait
  end for
  MPI_Reduce (sbuff:local_measurement_quantities,...)
  }
  On MPI_root{
  MPI_Finalize
  compute global_measurement_quantities
  }
```

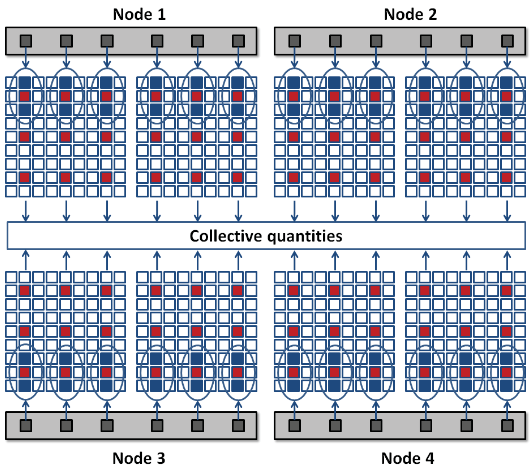


Figure 5. Hybrid MPI/OpenMP parallel application of the FSI algorithm for multiple Green's functions.

to compute the product of a matrix chain. At the wrapping step, $\widehat{G}$ are divided into b^2 seeds. Every OpenMP slave thread picks a seed and calculates its adjacent blocks until the selected inversion is formed. 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.

Note that the local measurement quantity calculations are carried out in the OpenMP 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 Sec.IV.

In addition, we note that since Green's functions need to be stored on all MPI processes temporarily for the calculation of measurements, the memory limitation of each node becomes one of major reasons to use a hybrid implementaiton rather than pure MPI.

IV. QMC simulation

At a high level, the DQMC simulation consists of two stages: warmup and physical measurement, see Alg. 4. A DQMC sweep in each stage travels through every site of the lattice in a multi-layer imaginary time slices, see Fig. 6.

Algorithm 4 DQMC simulation

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

DQMC sweep
for $\ell = 1, 2, ..., L$ do
    for $i = 1, 2, ..., N$ 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 $r \sim$ uniform$[0,1]$,
        if $r \leq \min\{1, r_{\ell i}\}$ then
            $h = h'$.
        end if
    end for
end for
```

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.

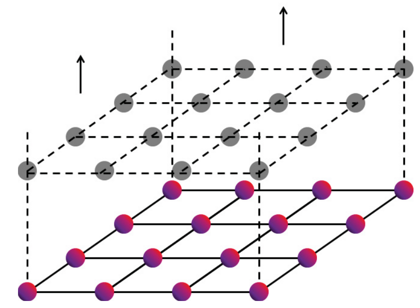


Figure 6. A $N = 4 \times 4$ lattice structure in a multi-layer imaginary time (L) slices.

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 [32], [33]. 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

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

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 $\{\mathrm{SPXX}(G^\sigma)\}_{(\tau,d)}$,

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

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 $\{\mathrm{SPXX}(G^\sigma)\}_{(\tau,d)}$, 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 $\{\mathrm{SPXX}(G^\sigma)\}_{(\tau,d)}$ are element-wise. It is extremely inefficient level-1 BLAS operations. FSI enables these calculations be executed in OpenMP multi-threads.

An overview of a full DQMC simulation with the highlights of the FSI algorithm for computing the Green's function and physicial measurements (PMs) 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

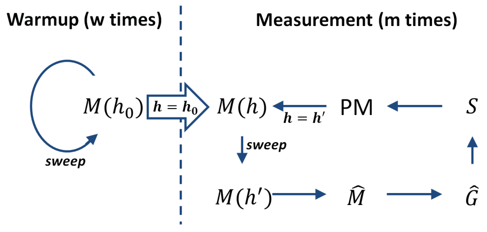


Figure 7. An overview of a full DQMC simulation with highlights of the FSI algorithm and physicial measurements (PMs).

FSI algorithm and its usage in a full DQMC simulation.

Our experiments were conducted on a Cray XC30 machine called 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 Intel "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 $0.25\mu s$ to $3.7\mu s$ MPI latency and 8 GB/sec MPI bandwidth.

A. Correctness validation

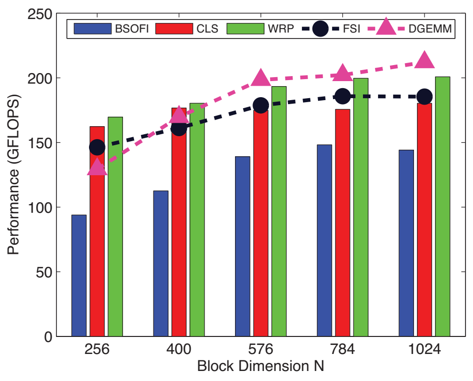
To validate the correctness of the FSI algorithm, we form a set of block p-cyclic Hubbard matrices M defined as (1). Each block B_ℓ is of the form $B_\ell = e^{t\Delta\tau K} e^{\sigma\nu V_\ell(h(\ell,:))}$, where $h = (h_{\ell,i}) = (\pm 1)$ for $1 \le \ell \le L$ and $1 \le i \le N$ are random variables, 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; $V_\ell(h(\ell,:)) = \text{diag}(h(\ell,1), h(\ell,2), \ldots, h(\ell,N))$.

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 approximately 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

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

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



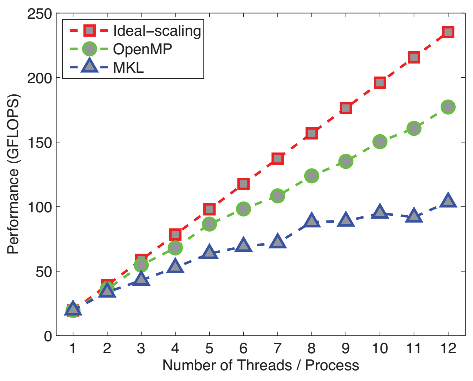


Figure 8. FSI performance rate (top) and scalability (bottom) on a single 12-core Intel "Ivy Bridge" processor.

profile of three steps of OpenMP multi-threaded FSI on the Intel "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.

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

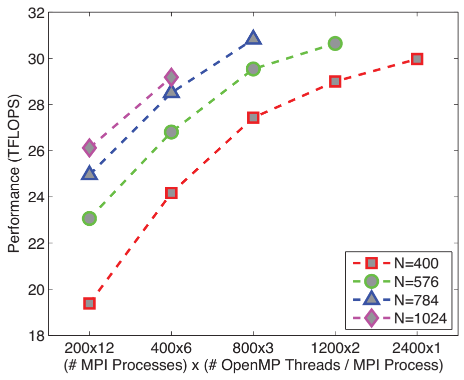


Figure 9. Performance rate of parallel application of FSI for multiple Green's functins with difference numbers of MPI processes and OpenMP threads.

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 one 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 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 the one of DGEMM, the peak rate in practice. In addition, FSI is scalable to the number of OpenMP threads and almost doubles the performance of pure multi-threaded MKL routines for computing a 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 Hubbard matrices $M(h)$ of the dimension $(L, N) = (100, 400)$. The cluster size $c = 10$. For both the equal-time and time-dependent measurements, we compute all diagonal blocks, b block rows and b block columns of each G. We compare the CPU time of serial execution, parallel executions with OpenMP and pure MKL, respectively, on an "Ivy Bridge" processor of Edition. Fig. 10

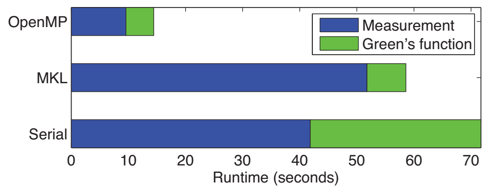


Figure 10. Runtime profile on a single Hubbard matrix.

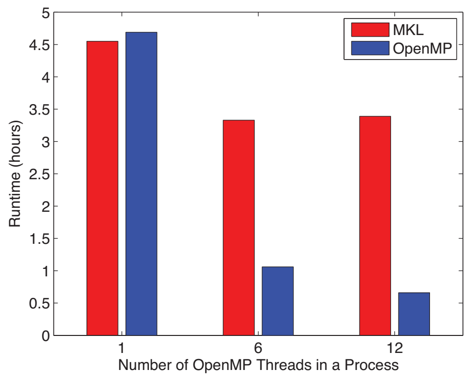


Figure 11. Runtime of a full DQMC simulation with $(w, m) = (100, 200)$ on an "Ivy Bridge" processor of Edition.

shows a profile of CPU time of computing the Green's function and physical measurements. As we can see, the pure MKL execution reduces the CPU time for computing Green's function due to the power of multi-threaded optimized LAPACK routine, but increases the CPU time for the physical measurements due to the execution of a sequential code in multi-threads. However, FSI with OpenMP uses 87% less CPU time for the computation of Green's functions and physical measurements.

Finally, we examine 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 runtime of the DQMC with FSI on an "Ivy Bridge" processor of Edition. As we can see, FSI with OpenMP gains a factor of 6.9 speedup from single-core to 12-core execution. In contrast, FSI with MKL only gains a factor of 1.3 speedup. As a result, the full DQMC simulation reduces from three and a half hours to forty minutes.

VI. Conclusion and Future work

In this paper, we tackled the bottleneck of Green's function calculations and physical measurements in DQMC simulations. The performance of the FSI algorithm has doubled the performance of simple Intel MKL calls. The enhancement of QMC capabilities by our work, will allow solution of problems that require either larger numbers of electrons or more complicated types of interactions.

One promising future work is the extension of the basic idea of the FSI algorithm to other types of structured matrices such as block tridiagonal matrices. Other future work includes a GPU implementation of the FSI and the hybrid massive parallelization of the full DQMC simulation.

ACKNOWLEDGEMENT. This research used resources of the NERSC, 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. CJ and ZB were supported in part by NSF grant CCF-1527091. RTS was supported in part by DOE grant DE-NA0002908.

REFERENCES

[1] R. Blankenbecler, D. J. Scalapino, and R. L. Sugar, "Monte Carlo calculations of coupled boson-fermion systems. i," *Phy. 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] G. Alvarez, M. S. Summers, D. E. Maxwell, M. Eisenbach, J. S. Meredith, J. M. Larkin, J. Levesque, T. A. Maier, P. R. C. Kent, E. F. D'Azevedo, and T. C. Schulthess, "New algorithm to enable 400+ TFlop/s sustained performance in simulations of disorder effects in high-T_c superconductors," *in Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pp. 61, 2008.

[4] 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.

[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 Green'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, "Selinv—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] S. Vatsya and H. Pritchard, "An explicit inverse of a tridiagonal matrix," *International journal of computer mathematics*, vol. 14, pp. 295–304, 1983.

[11] C. Da Fonseca and J. Petronilho, "Explicit inverse of a tridiagonal k-Toeplitz matrix," *Numerische Mathematik*, vol. 100, pp. 457–482, 2005.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] D. P. Woodruff, "Sketching as a tool for numerical linear algebra," *arXiv preprint arXiv:1411.4357*, 2014.

[18] 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.

[19] O. G. Ernst, "Equivalent iterative methods for p-cyclic matrices," *Numerical Algorithms*, vol. 25, pp. 161–180, 2000.

[20] G. Fairweather and I. Gladwell, "Algorithms for almost block diagonal linear systems," *SIAM Rev.*, vol. 46, pp. 49–58, 2004.

[21] W. J. Stewart, *Introduction to the numerical solution of Markov chains*. Princeton, NJ: Princeton Univ. Press, 1994, vol. 41.

[22] 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.

[23] 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.

[24] W. Gander and G. H. Golub, "Cyclic reduction – history and applications," *Scientific Computing*, pp. 73–85, 1997.

[25] J. E. Hirsch, "Stable Monte Carlo algorithm for fermion lattice systems at low temperatures," *Phy. Rev. B*, vol. 38, p. 12023, 1988.

[26] Z. Bai, W. Chen, R. Scalettar, and I. Yamazaki, "Numerical methods for quantum Monte Carlo simulations of the Hubbard model," in *Multi-Scale Phenomena in Complex Fluids*, 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] G. Hager and G. Wellein, *Introduction to high performance computing for scientists and engineers*. Boca Raton, FL: CRC Press, Taylor and Francis Group, 2010.

[32] N. W. Ashcroft and N. D. Mermin, *Solid State Physics*. Fort Worth: Cengage Learning, 1976.

[33] M. Tinkham, *Group theory and quantum mechanics*. Courier Dover Publications, 2003.