Numer. Math. Theor. Meth. Appl.
doi: 10.4208/nmtma.2009.m9008s

Vol. **2**, No. 4, pp. 469-484
November 2009

A High-Quality Preconditioning Technique for Multi-Length-Scale Symmetric Positive Definite Linear Systems

Ichitaro Yamazaki[1,*], Zhaojun Bai[1], Wenbin Chen[2] and Richard Scalettar[3]

[1] *Department of Computer Science, University of California, Davis, CA 95616, USA.*
[2] *School of Mathematical Sciences, Fudan University, Shanghai 200433, China.*
[3] *Department of Physics, University of California, Davis, CA 95616, USA.*

Received 3 May 2009; Accepted (in revised version) 22 July 2009

Abstract. We study preconditioning techniques used in conjunction with the conjugate gradient method for solving multi-length-scale symmetric positive definite linear systems originating from the quantum Monte Carlo simulation of electron interaction of correlated materials. Existing preconditioning techniques are not designed to be adaptive to varying numerical properties of the multi-length-scale systems. In this paper, we propose a hybrid incomplete Cholesky (HIC) preconditioner and demonstrate its adaptivity to the multi-length-scale systems. In addition, we propose an extension of the compressed sparse column with row access (CSCR) sparse matrix storage format to efficiently accommodate the data access pattern to compute the HIC preconditioner. We show that for moderately correlated materials, the HIC preconditioner achieves the optimal linear scaling of the simulation. The development of a linear-scaling preconditioner for strongly correlated materials remains an open topic.

AMS subject classifications: 65F08, 65F10, 65F50, 81-08

Key words: Preconditioning, multi-length-scale, incomplete Cholesky factorization, quantum Monte Carlo simulation.

1. Introduction

We consider the solution of the linear system of equations

$$Ax = b, \tag{1.1}$$

where A is an $n \times n$ *multi-length-scale* symmetric positive definite (SPD) matrix and b is a given vector of length n, originating from the hybrid quantum Monte Carlo (HQMC)

*Corresponding author. *Email addresses:* ic.yamazaki@gmail.com (I. Yamazaki), bai@cs.ucdavis.edu (Z. Bai), wbchen@fudan.edu.cn (W. Chen), scalettar@physics.ucdavis.edu (R. Scalettar)

simulation of a Hubbard model for studying electron interaction in correlated materials [2,15]. The properties of the coefficient matrix A, such as dimensionality and conditioning, depend on a set of multi-length-scale parameters from the underlying physical system to be simulated.

Preconditioning is recognized as one of the most critical components of a robust and efficient iterative linear solver. In this paper, we focus our attention on the development of a high-quality incomplete Cholesky (IC) factorization based preconditioner used in conjunction with the conjugate gradient method for solving the multi-length-scale system (1.1). An IC factorization is of the form

$$A = RR^T + E, \tag{1.2}$$

where R is a sparse lower-triangular matrix with positive diagonals, and E is an error matrix. A variety of IC factorization based preconditioners R have been proposed [1, 5–7, 9–11, 13, 16, 18]. The performance of many of these preconditioners for solving (1.1) has been reported in [3]. We observed that when a large number of elements are discarded into the error matrix E to control the cost of computing R, the norm of the residual matrix $R^{-1}AR^{-T} - I$ increases significantly and the quality of the preconditioner is poor. To overcome these drawbacks, we propose combining the two most effective IC factorizations, namely the IC factorization with a global diagonal shift by Manteuffel [13] and the robust IC factorization by Kaporin [11]. The resulting factorization is referred to as a hybrid Incomplete Cholesky (HIC) factorization. The HIC can adaptively balance the cost and quality of preconditioner over a wide range of the multi-length-scale parameters of interest. We will present an algorithm to compute the HIC factorization. To efficiently accommodate the data access pattern of the proposed algorithm, we will introduce a sparse matrix storage format, which is an extension of the well-known compressed sparse column (CSC) sparse matrix storage format. We will present numerical results to demonstrate the adaptivity of the HIC preconditioner to varying multi-length-scale parameters. For moderately correlated materials, the HIC preconditioner based PCG solver achieves the optimal linear scaling of the simulation. This enables us to conduct the HQMC simulation for thousands of electrons.

The rest of this paper is organized as follows. In Section 2, we review the existing IC factorizations that are closely related to the HIC factorization proposed in this paper. In Section 3, we define the HIC factorization and present an algorithm to compute the factorization. In Section 4, we discuss an implementation of the HIC with a new sparse matrix storage scheme. After detailing the form of the multi-length-scale linear system (1.1) in Section 5, we present numerical results to demonstrate the effectiveness of the HIC preconditioner in Section 6. The concluding remarks are in Section 7.

2. Incomplete Cholesky factorizations

For a general SPD matrix A, the IC factorization of the form (1.2) could fail due to the occurrence of non-positive pivot, referred to as *pivot breakdown* [12]. The existence is proven only for some special classes of matrices [13, 14, 17]. Various IC factorizations

have been proposed to avoid the pivot breakdown, see [5] and references therein. In this section, we review two IC factorizations, which are closely related to the one proposed in the next section.

To avoid the pivot breakdown, one can first introduce a diagonal shift and then compute the IC factorization proposed by Manteuffel [13]:

$$A + \alpha_{\mathrm{d}} D = R_{\mathrm{d}} R_{\mathrm{d}}^T + S_{\mathrm{d}} + S_{\mathrm{d}}^T, \tag{2.1}$$

where α_{d} is a scalar, D is the diagonal part of A denoted as $D = \mathrm{diag}(A)$, and S_{d} is a strictly lower-triangular matrix. The sparsity of R_{d} is imposed by a drop tolerance σ_1 and the algorithm to compute the factorization (2.1) is referred to as an IC_{d} algorithm for short. The residual norm of the IC_{d} factorization (2.1) is given by

$$\begin{aligned} \|R_{\mathrm{d}}^{-1} A R_{\mathrm{d}}^{-T} - I\| &= \|R_{\mathrm{d}}^{-1} E_{\mathrm{d}} R_{\mathrm{d}}^{-T}\| \\ &\le \|R_{\mathrm{d}}^{-1}\|^2 \|E_{\mathrm{d}}\| \\ &\le \|R_{\mathrm{d}}^{-1}\|^2 \left(2\|S_{\mathrm{d}}\| + \alpha_{\mathrm{d}} \|D\|\right), \end{aligned} \tag{2.2}$$

where $E_{\mathrm{d}} = S_{\mathrm{d}} - \alpha_{\mathrm{d}} D + S_{\mathrm{d}}^T$ is the error matrix. If the shift α_{d} is chosen such that $A + \alpha_{\mathrm{d}} D$ is strictly diagonally dominant, then the existence of the IC_{d} factorization (2.1) is guaranteed [13]. However, this may require a large shift α_{d} and lead to a large residual norm. In practice, it is sufficient to choose the shift α_{d} satisfying

$$A + \alpha_{\mathrm{d}} D > S_{\mathrm{d}} + S_{\mathrm{d}}^T. \tag{2.3}$$

Since the magnitudes of the elements of S_{d} are in $O(\sigma_1)$, α_{d} can be chosen at the order of σ_1. There is no general strategy for finding an optimal choice of α_{d}, and as such is often determined by a trial-and-error strategy [4].

Note that in the IC_{d} residual norm (2.2), the error matrix norm $\|E_{\mathrm{d}}\|$ is amplified by the factor $\|R_{\mathrm{d}}^{-1}\|^2$. When the matrix A is ill-conditioned, $\|R_{\mathrm{d}}^{-1}\|$ is typically large. To reduce the amplification factor $\|R_{\mathrm{d}}^{-1}\|^2$ in (2.2), Kaporin [11] proposed to impose the error matrix of the structure

$$E_{\mathrm{r}} = R_{\mathrm{r}} F_{\mathrm{r}}^T + F_{\mathrm{r}} R_{\mathrm{r}}^T + S_{\mathrm{r}} - D_{\mathrm{r}} + S_{\mathrm{r}}^T,$$

where F_{r} and S_{r} are lower-triangular and strictly lower-triangular matrices with the elements discarded from R_{r} and F_{r}, respectively. The sparsity of R_{r} is imposed by a primary drop tolerance σ_1, and the sparsity of F_{r} is imposed by a secondary drop tolerance σ_2, where $\sigma_2 < \sigma_1$. The diagonal matrix D_{r} is dynamically chosen such that

$$D_{\mathrm{r}} \ge S_{\mathrm{r}} + S_{\mathrm{r}}^T. \tag{2.4}$$

Hence the IC factorization proposed by Kaporin is of the form

$$A = R_{\mathrm{r}} R_{\mathrm{r}}^T + R_{\mathrm{r}} F_{\mathrm{r}}^T + F_{\mathrm{r}} R_{\mathrm{r}}^T + S_{\mathrm{r}} - D_{\mathrm{r}} + S_{\mathrm{r}}^T, \tag{2.5}$$

and can be equivalently written as

$$A + F_{\mathrm{r}} F_{\mathrm{r}}^T - S_{\mathrm{r}} + D_{\mathrm{r}} - S_{\mathrm{r}}^T = \left(R_{\mathrm{r}} + F_{\mathrm{r}}\right)\left(R_{\mathrm{r}} + F_{\mathrm{r}}\right)^T.$$

Since D_{r} satisfies (2.4), the existence of (2.5) is guaranteed. For this reason, the algorithm to compute the factorization (2.5) is referred to as a *robust* IC (RIC) algorithm.

In practice, when a nonzero value is assigned to the (i,j)-th element s_{ij} of S_{r}, to satisfy the condition (2.4), we first select positive scalars δ_i and δ_j such that $\delta_i\delta_j \geq s_{ij}^2$, and then increment the corresponding ith and jth diagonal elements d_i and d_j of D_{r} by δ_i and δ_j, respectively. For example, we can set $\delta_i = \tau_{ij}(a_{ii}+d_i)$ and $\delta_j = \tau_{ij}(a_{jj}+d_j)$ with

$$\tau_{ij} = \frac{|s_{ij}|}{\sqrt{(a_{ii}+d_i)(s_{jj}+d_j)}},$$

as proposed in [1].

The RIC residual norm is given by

$$\begin{aligned}\left\|R_{\mathrm{r}}^{-1}AR_{\mathrm{r}}^{-T}-I\right\| &= \left\|F_{\mathrm{r}}^{T}R_{\mathrm{r}}^{-T}+R_{\mathrm{r}}^{-1}F_{\mathrm{r}}+R_{\mathrm{r}}^{-1}(S_{\mathrm{r}}-D_{\mathrm{r}}+S_{\mathrm{r}}^{T})R_{\mathrm{r}}^{-T}\right\| \\ &\leq 2\|R_{\mathrm{r}}^{-1}\|\|F_{\mathrm{r}}\|+\|R_{\mathrm{r}}^{-1}\|^2\big(2\|S_{\mathrm{r}}\|+\|D_{\mathrm{r}}\|\big). \end{aligned} \tag{2.6}$$

Note that the magnitudes of the elements of F_{r} are in the order of the primary drop tolerance σ_1, and the amplification factor is reduced to be $\|R_{\mathrm{r}}^{-1}\|$, in comparsion to the factor $\|R_{\mathrm{d}}^{-1}\|^2$ in the IC_{d} residual norm (2.2). The second-order amplification factor $\|R_{\mathrm{d}}^{-1}\|^2$ only affects the term controlled by the secondary drop tolerance σ_2. Since $\sigma_2 \ll \sigma_1$ for some applications, RIC algorithm has demonstrated superior quality over IC_{d} algorithm and other IC-based algorithms [11].

Unfortunately, when solving the multi-length-scale linear system (1.1), we observe that a large number of nonzero values needs to be discarded to S_{r} for controlling the computational cost and storage requirement. As a result, some elements of D_{r} become large to ensure the condition (2.4). Subsequently, the term $\|R_{\mathrm{r}}^{-1}\|^2\|D_{\mathrm{r}}\|$ becomes dominant in the upper bound (2.6). If we want to keep it in the same order as the term $\|R_{\mathrm{r}}^{-1}\|\|F_{\mathrm{r}}\|$ in (2.6), the secondary drop tolerance σ_2 need to be set significantly smaller than σ_1. This leads to a large number of nonzeros in F_{r}. The computational cost and storage requirement of the RIC algorithm become significantly more than those of the IC_{d} algorithm.

3. Hybrid incomplete Cholesky factorization

In this section, we propose a hybrid algorithm of IC_{d} and RIC, which takes advantages of the flexibility in using the small diagonal shift α_{d} in the IC_{d} algorithm, and the small amplification factor $\|R_{\mathrm{r}}^{-1}\|$ in the RIC residual norm (2.6). Specifically, we propose an algorithm to compute an IC factorization of the form

$$A+\alpha D = RR^{T}+RF^{T}+FR^{T}+S+S^{T}, \tag{3.1}$$

where α is a scalar, $D=\mathrm{diag}(A)$, R is a lower-triangular matrix with positive diagonals, and F and S are strictly lower-triangular matrices. Furthermore, R, F, and S satisfy the following properties:

1) If $(i,j) \notin \mathscr{Z}_1$, then $r_{ij} = 0$;

2) If $(i,j) \notin \mathscr{Z}_2$, then $f_{ij} = 0$;

3) If $(i,j) \in \mathscr{Z}_1$ or $\mathscr{Z}_2$, then $s_{ij} = 0$.

Here $\mathscr{Z}_1$ is a set of ordered pairs (i,j) of integers containing at least all the pairs (i,i), and $\mathscr{Z}_2$ is a set of ordered pairs (i,j) such that $\mathscr{Z}_1 \cap \mathscr{Z}_2 = \emptyset$, i and $j \in \{1,2,\cdots,n\}$ and $i \geq j$. The properties 1) and 2) indicate that the elements of R and F are nonzero only on $\mathscr{Z}_1$ and $\mathscr{Z}_2$, respectively. Property 3) implies that the elements of S are zero on $\mathscr{Z}_1$ and $\mathscr{Z}_2$. Since $\mathscr{Z}_1 \cap \mathscr{Z}_2 = \emptyset$, the matrices R and F are called structurally orthogonal, that is, $r_{ij} f_{ij} = 0$ for all pairs (i,j). Furthermore, S is structurally orthogonal to both R and F.

Special cases of (3.1) include the case where $\alpha = 0$ and the set $\mathscr{Z}_1$ contains all the ordered pairs (i,j), $i \geq j$, then $F = S = 0$, and the factorization (3.1) becomes the Cholesky factorization. If $\mathscr{Z}_2 = \emptyset$, then $F = 0$ and the factorization (3.1) is the $\mathrm{IC_d}$ factorization (2.1). If the diagonal matrix αD is replaced by a diagonal matrix D_r and D_r is dynamically updated to ensure the condition (2.4), then the factorization (3.1) is the RIC factorization (2.5). Therefore, we refer to the algorithm to compute the factorization of the form (3.1) as a *hybrid IC (HIC) algorithm*.

The HIC algorithm to compute the factorization (3.1) can be derived by comparing the jth columns on both sides of (3.1):

$$(1+\alpha)a_{jj} = \sum_{k=1}^{j} r_{jk}^2,$$

$$a_{ij} = s_{ij} + \sum_{k=1}^{j} \left(r_{jk} r_{ik} + r_{jk} f_{ik} + f_{jk} r_{ik}\right), \quad \text{for } i = j+1,\cdots,n,$$

which can be equivalently written as

$$r_{jj}^2 = (1+\alpha)a_{jj} - \sum_{k=1}^{j-1} r_{jk}^2,$$

$$r_{jj}(r_{ij} + f_{ij}) + s_{ij} = a_{ij} - \sum_{k=1}^{j-1} \left(r_{jk} r_{ik} + r_{jk} f_{ik} + f_{jk} r_{ik}\right), \quad \text{for } i = j+1,\cdots,n.$$

Hence, to compute the jth column r_j of R, one first computes the vector v by updating the jth column a_j of A with the computed columns $r_1, r_2, \cdots, r_{j-1}$ of R and $f_1, f_2, \cdots, f_{j-1}$ of F:

$$v = a_j - \sum_{k=1}^{j-1} \left(r_{jk} r_k + r_{jk} f_k + f_{jk} r_k\right).$$

If the shift α is chosen such that $v_j + \alpha a_{jj} > 0$, then the diagonal element r_{jj} of R is given by

$$r_{jj} = \sqrt{v_j + \alpha a_{jj}}.$$

The rest of the elements of r_j, the elements of the jth column f_j of F and the jth column s_j of S take the form

$$\text{for } i = j+1, j+2, \cdots, n, \quad \begin{cases} r_{ij} = \frac{v_i}{r_{jj}}, & f_{ij} = 0, & s_{ij} = 0, & \text{if } (i,j) \in \mathscr{Z}_1, \\ r_{ij} = 0, & f_{ij} = \frac{v_i}{r_{jj}}, & s_{ij} = 0, & \text{if } (i,j) \in \mathscr{Z}_2, \\ r_{ij} = 0, & f_{ij} = 0, & s_{ij} = v_i & \text{otherwise.} \end{cases}$$

The following pseudocode implements this algorithm to compute the R-factor, where the sets $\mathscr{Z}_1$ and $\mathscr{Z}_2$ are defined using the prescribed drop tolerances σ_1 and σ_2 and $\sigma_2 < \sigma_1$:

HIC algorithm

for $j = 1, 2, \cdots, n$ **do**
  $v(j:n) = a(j:n, j)$
  $v(j) := (1+\alpha)v(j)$
  for $k = 1, 2, \cdots, j-1$ **do**
    $v(j:n) := v(j:n) - r(j,k)r(j:n,k)$
    $v(j:n) := v(j:n) - r(j,k)f(j:n,k)$
    $v(j:n) := v(j:n) - f(j,k)r(j:n,k)$
  end for
  if $v(j) \le 0$ **then** *pivot breakdown*
  $r(j,j) = \sqrt{v(j)}$
  for $i = j+1, j+2, \cdots, n$ **do**
    if $|v(i)|/r(j,j) > \sigma_1$ **then**
      $r(i,j) = v(i)/r(j,j)$ **and** $f(i,j) = 0$
    else if $|v(i)|/r(j,j) > \sigma_2$ **then**
      $f(i,j) = v(i)/r(j,j)$ **and** $r(i,j) = 0$
    else
      $r(i,j) = 0$ **and** $f(i,j) = 0$
    end if
  end for
end for

Note that the jth column r_j of R is computed through referencing the computed columns $r_1, r_2, \cdots, r_{j-1}$, which are on the left of r_j. Therefore, the HIC algorithm is referred to as a *left-looking* (columnwise) implementation. The computed columns $f_1, f_2, \cdots, f_{j-1}$ of F (which have to be stored explicitly) are referenced to update v. In Section 4, we will introduce a sparse matrix storage data structure for storing R and F for efficiently accommodating the access pattern of the HIC algorithm.

Alternatively, there is a *right-looking* implementation, in which the columns r_{j+1}, r_{j+2}, $\cdots$, r_n are updated by column r_j. This implementation discards the jth column of F once the columns r_{j+1}, r_{j+2}, $\cdots$, r_n are updated with it, thus saving storage costs. However, updating the columns $r_{j+1}, \cdots, r_n$ in a sparse data format is computationally expensive. A performance comparsion between the right-looking and left-looking implementations is in [19].

We now turn to compare the quality of HIC (3.1) and $\mathrm{IC_d}$ (2.1). Let us first compare the choice of diagonal shifts α and α_{d} and the corresponding residual norms. The HIC factorization (3.1) can be equivalently written as

$$A + FF^T - S + \alpha D - S^T = (R+F)(R+F)^T. \tag{3.2}$$

For the existence of the factorization, the shift α needs to be chosen such that

$$A + FF^T + \alpha D > S + S^T. \tag{3.3}$$

Note that the magnitudes of the elements of S are in the order of the secondary drop tolerance σ_2. Therefore, α can be chosen at the order of σ_2. Recall that the $\mathrm{IC_d}$ shift α needs to be $\mathcal{O}(\sigma_1)$ for satisfying (2.3).

The residual norm of the HIC factorization (3.2) is bounded by

$$\begin{aligned} \left\| R^{-1}AR^{-T} - I \right\| &= \left\| F^T R^{-T} + R^{-1}F + R^{-1}(S - \alpha D + S^T)R^{-T} \right\| \\ &\le 2\|R^{-1}\|\|F\| + \|R^{-1}\|^2(2\|S\| + \alpha\|D\|). \end{aligned} \tag{3.4}$$

The amplification factor $\|R^{-1}\|^2$ affects the term of the order σ_2. In the residual norm (2.2), the amplification factor $\|R^{-1}\|^2$ affects the term of the order σ_1. Since $\sigma_2 \le \sigma_1$, the HIC residual norm (3.4) is generally smaller than the $\mathrm{IC_d}$ residual norm (2.2).

Next, let us compare HIC (3.1) with RIC (2.5). As discussed in Section 2, the diagonal matrix D_{r} of RIC is dynamically updated to ensure the condition (2.4). The magnitudes of the elements of S_{r} are in the order of σ_2, which is in the same order as S of HIC (3.1). However, due to the presence of the positive definite matrix $A + FF^T$ in (3.3), the elements of αD that satisfy the condition (3.3) can be chosen to be smaller than those of D_{r} in order to satisfy (2.4). This indicates that the HIC residual norm (3.4) is generally smaller than the RIC residual norm (2.6).

4. Implementation issues

In this section, we propose a sparse matrix storage format to accommodate the data access pattern of the HIC algorithm described in Section 3. We note that on lines 4 to 8 of the HIC algorithm, the nonzeros of the matrices R and F are accessed to compute the array v. Specifically, v is first updated with the kth columns r_k of R and f_k of F for each nonzero r_{jk} in the jth row of R, and then v is updated with r_k for each nonzero f_{jk} in the jth row of F. To accommodate this specific data access pattern, we propose a sparse matrix storage format to simultaneously store a pair of matrices of the same dimension.

Let us consider two matrices A and B of dimension $m \times n$ and denote the numbers of nonzeros in A and B by nnz_A and nnz_B, respectively. Then, we store A and B in a set of six arrays, respectively:

$$\begin{array}{l} \texttt{valA, rowA, ptrA, linkA, headA, nextB,} \\ \texttt{valB, rowB, ptrB, linkB, headB, nextA,} \end{array} \tag{4.1}$$

where

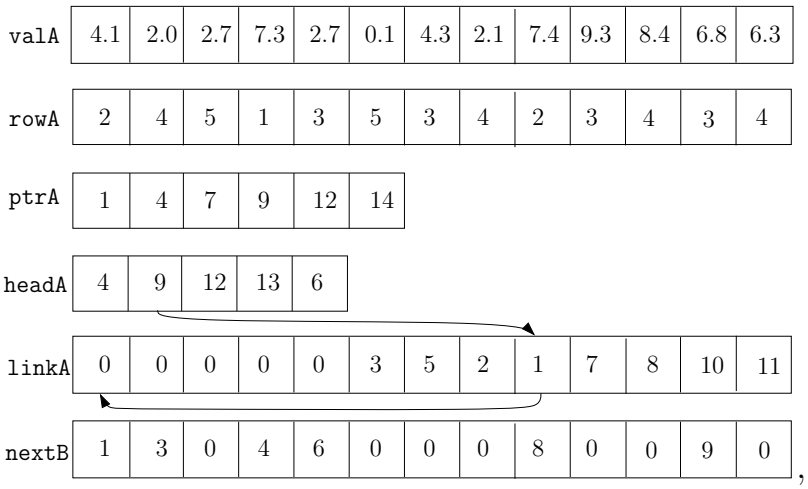
- `valA` is a real array of length nnz_A that stores the values of the nonzeros of A as they are traversed in a columnwise fashion.
- `rowA` is an integer array of length nnz_A that stores the row indexes of the nonzeros in `valA`. If $\texttt{valA}(k) = a_{ij}$, then $\texttt{rowA}(k) = i$.
- `ptrA` is an integer array of length $n+1$. $\texttt{ptrA}(j)$ points to the location of `valA` that stores the first nonzero in the jth column of A. By convention, $\texttt{ptrA}(n+1) = 1 + nnz_A$.
- `headA` is an integer array of length n. $\texttt{headA}(i)$ specifies the location of `valA` that stores the right-most nonzero in the ith row of A, which is referred to as the head of the ith row.
- `linkA` is an integer array of length nnz_A that forms a linked-list of the nonzeros in the same row of A. Specifically, if $\texttt{valA}(k) = a_{ij}$, then $\texttt{linkA}(k)$ specifies the location of `valA` that stores the nonzero on the immediate left of a_{ij} in the ith row of A. If such a nonzero does not exist, then $\texttt{linkA}(k) = 0$.
- `nextB` is an integer array of length nnz_A that links a nonzero of A with a nonzero of B. Specifically, if $\texttt{valA}(k) = a_{ij}$, then $\texttt{nextB}(k)$ specifies the location of `valB` that stores the first nonzero of B below the ith row in the jth column. If such a nonzero does not exist, then $\texttt{nextB}(k) = 0$.
- `valB`, `rowB`, `ptrB`, `headB`, `linkB`, and `nextA` for B are defined in the same way.

Note that the first three arrays `valA`, `rowA` and `ptrA` define a storage format commonly known as the compressed sparse column (CSC) format [8]. The extension of CSC format by adding the fourth and fifth arrays `headA` and `linkA` was proposed in [1], which we refer to as a CSC with row access (CSCR) format. The storage format (4.1) to simultaneously store a pair of matrices A and B in CSCR with additional linkage arrays `nextA` and `nextB` is an extension of CSCR (CSCRe).

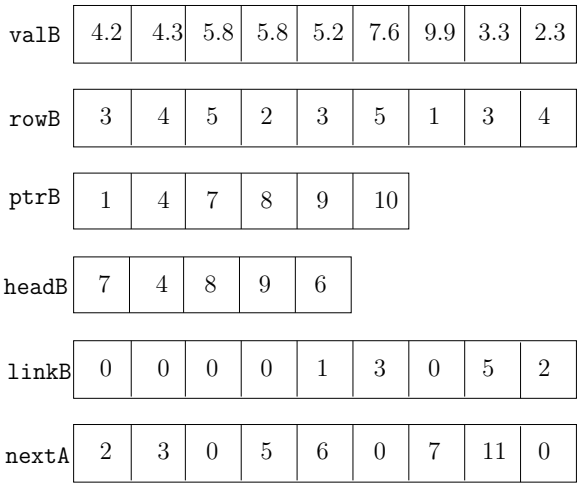
Example. Consider the following pair of 5×5 matrices A and B:

$$A = \begin{bmatrix} & 7.3 & & & \\ 4.1 & & & 7.4 & \\ & 2.7 & 4.3 & 9.3 & 6.8 \\ 2.0 & & 2.1 & 8.4 & 6.3 \\ 2.7 & 0.1 & & & \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} & & 9.9 & & \\ & 5.8 & & & \\ 4.2 & 5.2 & & 3.3 & \\ 4.3 & & & & 2.3 \\ 5.8 & 7.6 & & & \end{bmatrix}. \tag{4.2}$$

By the CSCRe format (4.1), the matrix A is stored as



and the matrix B is stored as



After the matrices A and B are stored in the CSCRe format, we can see that the following data access patterns can be directly accommodated: (a) The nonzeros in the same column of A can be accessed in ascending order of row indexes using `valA`, `rowA`, and `ptrA`. For example, the nonzeros 4.1, 2.0, and 2.7 in the first column of A can be retrieved from `valA` at the contiguous locations starting at $\texttt{ptrA}(1) = 1$ and ending at $\texttt{ptrA}(2) - 1 = 3$. (b) The nonzeros in the same row of A can be accessed using `valA`, `headA`, and `linkA`. For example, to retrieve the nonzeros 7.4 and 4.1 in the second row of the matrix A, note first that $\texttt{headA}(2) = 9$. Thus, the nonzero 7.4 in the second row is stored in $\texttt{valA}(9)$. Since $\texttt{linkA}(9) = 1$, the next nonzero 4.1 in the same row is stored in $\texttt{valA}(1)$. Finally, $\texttt{linkA}(1) = 0$, indicating that all the nonzeros in the row have been retrieved. (c) The nonzeros b_{ij} of B below the kth row in the same column as a nonzero a_{kj} of A can be accessed in ascending order of row indexes using `nextB`, `ptrB`, `valB`, and `rowB`. For example, consider the nonzero $a_{21} = 4.1$ of A which is stored in $\texttt{valA}(1)$. Then, the nonzeros 4.2, 4.3, and 5.8 of B below the second row in the first column can be retrieved

from valB at the contiguous locations starting at $\texttt{nextB}(1) = 1$ and ending at $\texttt{ptrB}(2) - 1 = 3$. Similarly, since the nonzero $b_{13} = 9.9$ of B is stored in $\texttt{valB}(7)$, the nonzeros 4.3 and 2.1 of A below the first row in the third column can be read from valA at the locations starting at $\texttt{nextA}(7) = 7$ and ending at $\texttt{ptrA}(4) - 1 = 8$.

Let us now consider how to implement the key steps of the HIC algorithm using the CSCRe format (4.1). At the line 6, the array v is updated with the kth column f_k of F for each nonzero r_{jk} in the jth row of R. This is implemented by the following subroutine, where the computed columns of R and F are stored in the CSCRe format.

```
UPDATE(j, nnz_v, indV, v, valR, headR, linkR, nextF, valF, rowF)
 1. ℓ = headR(j)
 2. while ℓ ≠ 0
 3.     t = nextF(ℓ)
 4.     while rowF(t) > j
 5.         if v(rowF(t)) = 0 then
 6.           nnz_v := nnz_v + 1
 7.           indV(nnz_v) := rowF(t)
 8.         end if
 9.         v(rowF(t)) := v(rowF(t)) − valR(ℓ)valF(t)
10.         t := t + 1
11.     end while
12.     ℓ = linkR(ℓ)
13. end while
```

In the above subroutine, the nonzeros r_{jk} in the jth row of R are accessed using headR, valR, and linkR (lines 1, 9, and 12). Then, the nonzeros of F in the kth column below the jth row are accessed using nextF, valF, and rowF; the location of valF that stores the first nonzero is specified by nextF (line 3), and the rest of the nonzeros are retrieved from valF at the contiguous locations thereafter (lines 4 to 11). To identify the last nonzero in the kth column of F, the diagonal elements of F are stored even though they are all zero. Hence, when the row index is less than or equal to j (line 4), the nonzero corresponds to the $(k+1)$th diagonal element of F. Note that ptrF is not used.

On line 7 of the HIC algorithm, the array v is updated with the kth column r_k of R for each nonzero element f_{jk} in the jth row of F. This is done by calling the subroutine UPDATE with the arguments (j, nnz_v, indV, v, valF, headF, linkF, nextR, valR, rowR). The line 5 of the algorithm can be implemented in a similar fashion.

On line 13 of the HIC algorithm, if the magnitude of $\texttt{v}(i)$ is greater than a prescribed primary drop tolerance σ_1, then it is assign $\texttt{v}(i)$ to the element r_{ij} of R. The following subroutine saves r_{ij} by inserting r_{ij} into the data structure.

INSERT(i, v, nnz_R, valR, rowR, headR, linkR, prv_F, nnz_F, nextR)

1. $nnz_R := nnz_R + 1$
2. valR(nnz_R) = v(i)
3. rowR(nnz_R) = i
4. linkR(nnz_R) = headR(i)
5. headR(i) = nnz_R
6. **for** $k = prv_F, prv_F + 1, \cdots, nnz_F$
7. nextR(k) = nnz_R
8. **end for**
9. $prv_F = nnz_F + 1$

A few remarks are in order. (a) On lines 1 to 3, v(i) is stored as the element r_{ij} of R. (b) On lines 4 and 5, the element r_{ij} is inserted into the linked-list of the nonzero elements in the ith row of R. (c) Let r_{kj} be the last nonzero element assigned to the jth column of R before the element r_{ij}. Then, prv_F in the arguments specifies the location of valF that stores the first nonzero element of F below the kth row in the jth column. On lines 6 to 8, prv_F is used to link the nonzero elements of F to the nonzero element r_{ij} of R.

If the magnitude of $v(i)$ on line 15 of the HIC algorithm is less than or equal to σ_1 but greater than a prescribed secondary drop tolerance σ_2, then v(i) is assigned to the element f_{ij} of F. This is done by calling the subroutine INSERT with the arguments (i, v, nnz_F, valF, rowF, headF, linkF, prv_R, nnz_R, nextF).

5. Multi-length-scale linear systems

In this section, we describe the multi-length-scale SPD linear systems of equations originating from the hybrid quantum Monte Carlo (HQMC) simulation of the Hubbard model, a powerful tool for studying the electron interactions that characterize the magnetic and transport properties of correlated materials [15]. The numerical solution of such multi-length-scale linear systems is a computational bottleneck in the HQMC simulation [2, 3]. Specifically, the coefficient matrix A of the multi-length-scale system (1.1) is defined as

$$A = \begin{bmatrix} I + B_2^T B_2 & -B_2^T & & & B_1 \\ -B_2 & I + B_3^T B_3 & -B_3^T & & \\ & \ddots & \ddots & \ddots & \\ & & -B_{L-1} & I + B_L^T B_L & B_L^T \\ B_1^T & & & -B_L & I + B_1^T B_1 \end{bmatrix}, \tag{5.1}$$

where for $\ell = 1, 2, \cdots, L$, B_ℓ is an $N \times N$ matrix defined by the product

$$B_\ell = B e^{D_\ell}.$$

B is an $N \times N$ matrix given by the Kronecker product:

$$B = \left(B_m^{(1)} B_m^{(2)}\right) \otimes \left(B_m^{(1)} B_m^{(2)}\right),$$

and $B_m^{(1)}$ and $B_m^{(2)}$ are $m \times m$ matrices defined as

$$B_m^{(1)} = \begin{bmatrix} C & & & & \\ & C & & & \\ & & \ddots & & \\ & & & C & \\ & & & & C \end{bmatrix}, \quad B_m^{(2)} = \begin{bmatrix} \cosh\theta & & & & \sinh\theta \\ & C & & & \\ & & \ddots & & \\ & & & C & \\ \sinh\theta & & & & \cosh\theta \end{bmatrix},$$

where

$$C = \begin{bmatrix} \cosh\theta & \sinh\theta \\ \sinh\theta & \cosh\theta \end{bmatrix}$$

and θ is a scalar. The matrix D_ℓ of order N is diagonal:

$$D_\ell = \eta\, \mathrm{diag}(h^{(\ell)}) + \nu I,$$

where η and ν are scalars and $h^{(\ell)}$ is a random vector of length N. Therefore, the order of the matrix A is $n = N \times L = m \times m \times L$. The matrix A is extremely sparse, having about 68 nonzeros per column.

The integer N is the total number of the sites on an $m \times m$ rectangular spatial lattice, $N = m \times m$. The random vector $h^{(\ell)}$ is called the Hubbard-Stratonovich configuration. The scalars η, ν and θ are defined as

$$\eta = \gamma \Delta\tau (2U)^{\frac{1}{2}}, \quad \nu = \mu \Delta\tau, \quad \theta = t\Delta\tau,$$

where $\gamma = 1$ or -1, $\Delta\tau = \beta/L$. Scalars β, t, U and μ are parameters specifying the (inverse) temperature, kinetic energy, potential energy and chemical potential of the underlying physical system, respectively. The scalar L is the number of imaginary time-intervals. These system parameters, namely N, L, β, t, μ and U, define the properties of the underlying physical systems. The linear system (1.1) is referred to as a multi-length-scale linear system since the dimensionality, eigenvalue distribution and conditioning of the coefficient matrix A depends on the choice of the system parameters. Fig. 1 shows the condition numbers of A as a function of N, β and U. As we can see, the matrix A becomes extremely ill-conditioned as the values of β and U increase. This is the case for the so-called strongly-interacting system [3, 15].

6. Numerical results

In this section, we present numerical results of the preconditioned conjugate gradient method to solve the multi-length-scale linear systems (1.1) with the IC_d, RIC and HIC preconditiners. We will only present the performance data for the choice of parameters

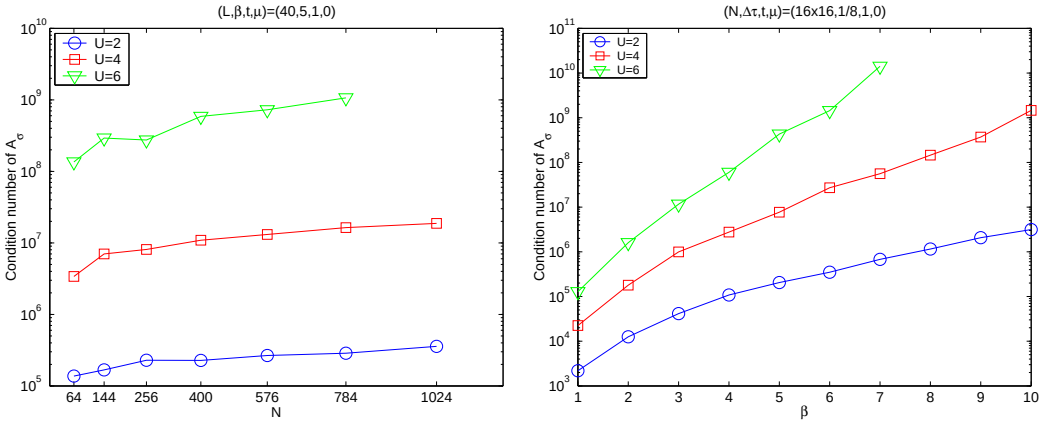


Figure 1: Condition numbers of the Hubbard matrix A as a function of N (left) and β (right).

$N = 48 \times 48$, $L = 80$, $\beta = 10$, $t = 1$, $\mu = 0$ and $U = 0, 1, 2, \cdots, 6$. The dimension of the linear system is $n = 48 \times 48 \times 80 = 184,320$. Numerical results for other sets of parameters can be found in [19]. To reflect the realistic HQMC simulation, the entries of the Hubbard-Stratonovich configuration $h^{(\ell)}$ is set to follow the Gaussian distribution with a mean of zero and standard deviation of two. The right-hand-side vector b is set such that elements of the exact solution vector x are uniformly distributed in the interval $[0, 1)$. For a fair comparison, all preconditioners were constructed to have about the same sparsity (about 25 nonzeros per row). This is accomplished by choosing the shift $\alpha_{\mathrm{d}} = 5 \times 10^{-3}$ and the drop tolerance $\sigma_1 = 5 \times 10^{-3}$ for IC_{d}, $(\sigma_1, \sigma_2) = (5.0 \times 10^{-3}, 2.5 \times 10^{-4})$ for RIC, and $\alpha = 7.0 \times 10^{-4}$ and $(\sigma_1, \sigma_2) = (7.0 \times 10^{-3}, 7.0 \times 10^{-4})$ for HIC.

The IC algorithms and PCG method discussed in this paper have been implemented in Fortran 95. The initial PCG iterate is set to be zero. The PCG iteration is declared to be convergent when the computed solution $\widehat{x}$ has three correct significant decimal digits, i.e., $\|x - \widehat{x}\|_2 / \|x\|_2 \leq 10^{-3}$. This is sufficient for the practical needs of the HQMC simulation. All data was collected on an HP Itanium2 workstation with 1.5GHz CPU and 2GB of main memory. Intel Math Kernel Library 7.2.1 and the `-O3` optimization option were used to compile the programs. Taking the effects of the randomness into the account, the reported data are the averages of fifty runs.

Let us first compare the PCG convergence rate by showning the numbers of iterations in the following table:

U	0	1	2	3	4	5	6
IC_{d}	14	32	72	190	1,087	3,795	5,400
RIC	12	29	66	106	1,026	3,683	5,412
HIC	35	28	51	132	685	2,029	2,978

From the table, we see that the HIC preconditioner makes significant improvement in terms of the convergence rate. We note that there is a large jump from $U = 3$ to $U = 4$, which is due to the fact that the coefficient matrices A are extremely ill-conditioned for large U.

This jump leads to the large amplification factors $\|R^{-1}\|$ in the IC_d, RIC and HIC residual norms.

As discussed in Section 3, the higher quality of the HIC preconditioner is due to the fact that the diagonal elements of the error matrix $A - RR^T$ are smaller than those of IC_d and RIC preconditioners. Note that the shift $\alpha = 7.0 \times 10^{-4}$ of HIC is about an order of magnitude smaller than the shift $\alpha_d = 5.0 \times 10^{-3}$ of IC_d. The following table indicates that the diagonal elements of the HIC error matrices are about two orders of the magnitude smaller than the ones of RIC.

U	0	1	2	3	4	5	6
HIC/RIC $(\alpha\|D\|_1)/\|D_r\|_1$	0.013	0.029	0.037	0.053	0.067	0.076	0.096
RIC/IC_d $\|D_r\|_1/(\alpha_d\|D\|_1)$	10.45	4.85	3.76	2.66	2.08	1.85	1.46

From the table, we also see that the orders of diagonal elements of RIC and IC_d are about the same. Hence, it takes about the same numbers of PCG iterations to converge using the RIC and IC_d preconditioners.

The CPU time of computing the IC_d preconditioners is between 1.23 to 1.32 seconds for different U. It is 5.46 to 6.36 seconds for the RIC preconditioners, and 3.24 to 4.23 seconds for the HIC preconditioners. The following table shows the total CPU elapsed time (in seconds) to solve the multi-length-scale linear systems (1.1):

U	0	1	2	3	4	5	6
IC_d	1.87	2.86	4.77	10.48	53.49	184.76	287.47
RIC	6.13	7.86	9.40	15.05	56.77	188.05	301.71
HIC	4.94	5.49	6.68	10.67	38.09	104.88	167.55

As we can see, the HIC preconditioners significantly reduce the total solution time for strongly-interacting system, namely $U \geq 4$. For $U \leq 3$, the total CPU time of HIC preconditioners is greater than those of IC_d preconditioners due to the higher cost of the HIC preconditioners. Note that the HIC construction time can be reduced by using larger drop tolereances. As discussed in Section 3, the HIC factorization is a generalization of the IC_d factorization, and optimal performance of the HIC is at least as good as that of the IC_d. Therefore, by adaptively adjusting the drop tolerances, the HIC can adapt to the varying mathematical properties of the multi-length-scale linear systems.

Finally, we examine the scaling property of the HIC-based PCG solver with respect to the parameter N of the dimensionality and the potential energy parameter $U = 0, 1, 2, 3$. The drop tolerances of the HIC factorization are chosen such that all R-factors have about 55 nonzero elements per row. The left plot of Fig. 2 shows that the numbers of PCG iterations only grows slowly as N increase. As a result, the solution time scales linearly with N (see the right plot of Fig. 2). In the figure, the dashed lines are the ideal linear scaling lines using the solution times taken at $N = 48 \times 48$ and $U = 1$ and 3. Hence, we conclude that the HIC-based PCG solver achieves the property of optimal linear-scaling. Unfortunately, the similar linear-scaling property does not hold when $U \geq 4$.

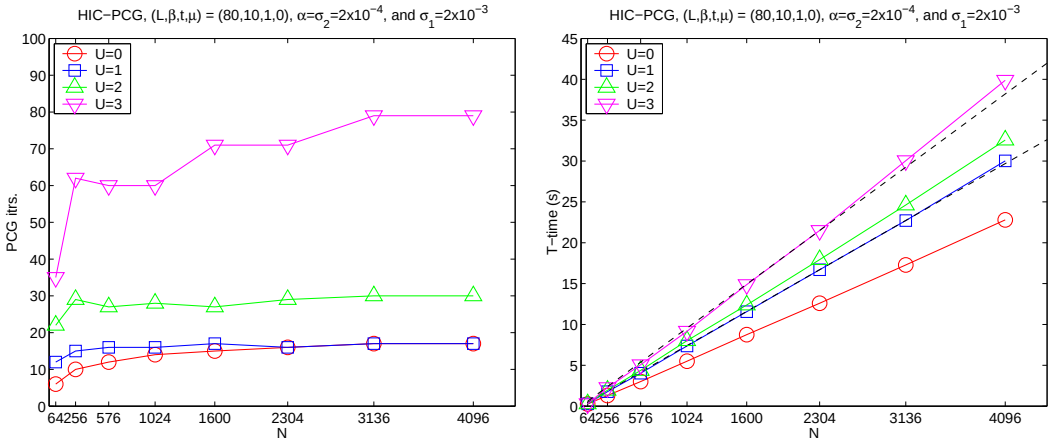


Figure 2: Left: the number of iterations. Right:the CPU elapsed time.

7. Conclusion

We have introduced a hybrid IC preconditioning algorithm to solve the multi-length-scale SPD linear systems (1.1) and (5.1). The numerical results have demonstrated that the HIC preconditioner is an effective preconditioner for solving the multi-length-scale SPD linear systems for a wide range of parameters of interest. It shows that for moderately interacting physical systems ($U < 3$), the HIC-based PCG linear solver scales linearly with respect to the dimensionality of the system. Thus it enables the HQMC simulation of thousands of moderately interacting electrons [19]. The development of a preconditioner that enables the optimal linear-scaling for strongly-interacting systems remains an open problem.

Acknowledgments The work was supported in part by the US National Science Foundation grant 0611548 and in part by the US Department of Energy grant DE-FC02-06ER25793. We thank the anonymous referees for their insightful comments.

References

[1] M. Ajiz and A. Jennings, *A robust incomplete Choleski-conjugate gradient algorithm*, Int. J. Numer. Meth. Engng., (1984), pp. 949–966.

[2] Z. Bai, W. Chen, R. Scalettar, and I. Yamazaki, *Robust and efficient numerical linear algebra solvers and applications in quantum mechnical simulations*, in Proc. The Fourth International Congress of Chinese Mathematician (ICCM), L. Ji, K. Liu, Y. Yang, and S. Yau, eds., 2007, pp. 253–268.

[3] Z. Bai, W. Chen, R. T. Scalettar, and I. Yamazaki, *Lecture notes on advances of numerical methods for quantum Monte Carlo simulations of the Hubbard model*, CSE-2007-36, Department of Computer Science, University of California, Davis, 2007.

[4] S. Balay, K. Buschelman, W. D. Gropp, D. Kaushik, M. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang, *PETSc User's Manual*, technical Report ANL-95/11 - Revision 2.1.5, Argonne National Laboratory, 2002.

[5] M. Benzi, *Preconditioning techniques for large linear systems: A survey*, J. Comput. Phys., 182 (2002), pp. 418–477.
[6] M. Benzi and M. Tuma, *A robust incomplete factorization preconditioner for positive definite matrices*, Numer. Linear Algebra Appl., 10 (2003), pp. 385 – 400.
[7] M. Benzi and M. Tuma., *A robust preconditioner with low memory requirements for large sparse least squares problems*, SIAM J. Sci. Comput., 25 (2003), pp. 499–512.
[8] I. S. Duff, *A survey of sparse matrix research*, in Proc. IEEE, vol. 65, 1977, pp. 500–535.
[9] V. Eijkhout, *The 'weighted modification' incomplete factorization method*, LAPACK Working Note 145, UT-CS-99-436, Computer Science Department, University of Tennessees, 1999.
[10] A. Jennings and G. M. Malik, *Partial elimination*, Journal of the J. Inst. Math. Appl., 20 (1977), pp. 307–316.
[11] I. E. Kaporin, *High quality preconditioning of a general symmetric positive definite matrix based on its $U^TU+U^TR+R^TU$-decomposition*, Numer. Linear Algebra Appl., 5 (1998), pp. 483–509.
[12] D. S. Kershaw, *The incomplete Cholesky-conjugate gradient method for the iterative solution of systems of linear equations*, J. Comput. Phys., 26 (1978), pp. 43 – 65.
[13] T. A. Manteuffel, *An incomplete factorization technique for positive definite linear systems*, Math. Comp., 34 (1980), pp. 473–497.
[14] J. Meijerinkand and H. A. van der Vorst, *An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix.*, Math. Comp., 31 (1977), pp. 134–155.
[15] R. T. Scalettar, D. J. Scalapino, R. L. Sugar, and D. Tousaint, *A hybrid-molecular dynamics algorithm for the numerical simulation of many electron systems*, Phys. Rev. B, 36 (1987), pp. 8632–8641.
[16] M. Tismenetsky, *A new preconditioning technique for solving large sparse linear systems*, Linear Algebra Appl., (1991), pp. 331–353.
[17] R. S. Varga, E. B. Saff, and V. Mehrmann, *Incomplete factorizations of matrices and connections with H-matrices*, SIAM J. Numer. Anal., 17 (1980), pp. 787–793.
[18] X. Wang, K. A. Gallivan, and R. Bramley, *CIMGS: An incomplete orthogonal factorization preconditioner*, SIAM J. Sci. Comput., 18 (1997), pp. 516–536.
[19] I. Yamazaki, *High-quality preconditioning techniques for multi-length-scale symmetric positive definite matrices and their applications to the hybrid quantum Monte Carlo simulation of the Hubbard model*, PhD thesis, University of California, Davis, 2008.