Computer Science • 19(4) 2018 https://doi.org/10.7494/csci.2018.19.4.2868

Yingchun Zhang
Quanyi Lv
Manyu Xiao
Gongnan Xie
Piotr Breitkopf

A PARALLEL ALGORITHM OF ICSYM FOR COMPLEX SYMMETRIC LINEAR SYSTEMS IN QUANTUM CHEMISTRY

Abstract *Computational effort is a common issue for solving large-scale complex symmetric linear systems, particularly in quantum chemistry applications. In order to alleviate this problem, we propose a parallel algorithm of improved conjugate gradient-type iterative (ICSYM). Using three-term recurrence relation and orthogonal properties of residual vectors to replace the tridiagonalization process of classical CSYM, which allows to decrease the degree of the reduce-operator from two to one communication at each iteration and to reduce the amount of vector updates and vector multiplications. Several numerical examples are implemented to show that high performance of proposed improved version is obtained both in convergent rate and in parallel efficiency.*

Keywords complex symmetric linear systems, parallel computing, improved conjugate gradient-type iterative algorithm (ICSYM)

Citation Computer Science 19(4) 2018: 385–401

1. Introduction

In practical quantum chemistry applications, a complex symmetric eigenvalue problem is encountered for determining the laser-induced molecular resonance states of H_2^+ in an electro-magnetic field [1]. The dominant computational time is to solve complex symmetric linear systems, particularly for determining several eigenvalues. Thus, in this work we concentrate on developing an efficient algorithm to solve this linear system, $Ax = b$, where A is an $n \times n$ non-Hermitian but symmetric matrix, i.e., $A \neq A^{\mathrm{H}}$ and $A = A^{\mathrm{T}} \in \mathbb{C}^{n\times n}$. In fact, this typical kind of linear systems also arise from other valuable application areas such as electromagnetic scattering from a large open cavity [12], scattering problems in computational electromagnetics [17], Maxwell's equations [8, 10] and Helmholtz equations [6, 9]. With the development of technology and methodology, more large-scale problems are encountered to be solved efficiently. Recently, advanced methodologies for solving complex symmetric linear systems have been thoroughly discussed in many papers, such as the complex symmetric quasi-minimal residual method (QMR-SYM) [14], symmetric complex bi-conjugate gradient conjugate residual-type method (SCBiCG) [5, 7], bi-conjugate gradient conjugate residual-type method (BiCGCR) [5, 7], conjugate orthogonal conjugate gradient method (COCG) [15] and conjugate A-orthogonal conjugate residual method (COCR) [7, 13]. However, they have a common limitation that they are not stable or applicable in large-scale dense complex symmetric linear systems. In order to overcome this limitation, the conjugate gradient-type iterative algorithm (CSYM) [3] was proposed, as it has tremendous advantages in small storage capacity and stable computations.

From the parallel algorithm of view, the above algorithms applied for solving large-scale linear systems suffer a main bottleneck due to the global communication of the inner products. It becomes more serious when the number of parallel processors is argumented. This results in much lower parallel efficiency.

Nowadays, there are three popular strategies to overcome this issue. The first one is to decrease the number of the reduce-operator by eliminating data dependencies. The second one is to restructure the algorithm so that the communication and computation can be overlapped efficiently. The last one is to replace the computation involving global communications by the other computation without global communications [18, 21]. Moreover, these strategies have been widely applied to develop the parallel algorithms for solving large linear systems arising from the practical engineering applications, such as the parallel QMR method [2], the improved conjugate residual squared method (ICRS) [20], the improved Biorthogonal Conjugate Gradient method (BiCG) [18], the improved stabilized BiCG method (BiCGSTAB) [19] and the parallel COCR method (PCOCR) [21]. However, the above methods exhibit poor parallel performance and numerical instabilities for solving relatively dense linear systems. Therefore, in this work, an improved parallel algorithms based on CSYM is proposed by adopting the above parallel strategies to solve large-scale complex symmetric linear systems, particularly more efficient for large-scale dense matrices.

The remainder of this paper is organized as follows. In Section 2, description of the improved CSYM algorithm is mentioned for solving linear systems with large relatively dense or dense complex symmetric matrices. Then, theoretical analysis and parallel implementation about two algorithms (CSYM and ICSYM) are presented in Section 3. In Section 4, numerical experiments arising in some practical problems are discussed. Finally, we end with some conclusions in Section 5.

For the convenience of our statements, we use the following notation throughout the paper: the symbols $\overline{A}$, A^{T} are used to denote the conjugate, the transpose, respectively. The inner product in space $\mathbb{C}^n$ is defined as $[x, y] = y^{\mathrm{H}}x$, where $x, y \in \mathbb{C}^n$ are column vectors. $\|x\|_2$ is the 2-norm of vector x.

2. Description of improved CSYM

In order to highlight the improved characteristics of ICSYM compared with classical CSYM, we firstly introduce a brief overview of the CSYM algorithm [3]. Then the detailed induction of ICSYM is depicted.

2.1. Fundamental of CSYM

Given an initial vector x_0, the residual vector $r_0 = b - Ax_0$, then $q_1 = \frac{\overline{r_0}}{\|r_0\|_2}$. In order to obtain $q_2, q_3, \ldots,$ three-term recurrence relation is iteratively implemented by the following scheme,

$$AQ_k = \overline{Q_k}H_k + b_k\overline{q_{k+1}}e_k^{\mathrm{T}}, \tag{1}$$

where $Q_k = (q_1, q_2, \ldots, q_k)$, $e_k = (0, 0, \ldots, 1)^{\mathrm{T}} \in \mathbb{R}^k$, $k = 1, 2, \ldots$ and H_k is a $k \times k$ symmetric tridiagonal matrix,

$$H_k = \begin{pmatrix} a_1 & b_1 & & & \\ b_1 & a_2 & b_2 & & \\ & b_2 & a_3 & \ddots & \\ & & \ddots & \ddots & b_{k-1} \\ & & & b_{k-1} & a_k \end{pmatrix}.$$

The iteration x_k can be obtained from the column space of Q_k,

$$x_k = x_0 + Q_k f_k \in span\{q_1, q_2, \ldots, q_k\}, \quad k = 1, 2, \ldots,$$

where $f_k = (f_k^{(1)}), f_k^{(2)}, \ldots, f_k^{(k)})^{\mathrm{T}} \in \mathbb{C}^k$. Since $r_k = b - Ax_k$ is orthogonal to the conjugate column vectors of Q_k denoted $(\overline{q_1}, \overline{q_2}, \ldots, \overline{q_k})$, i.e.

$$[r_k, \overline{q_i}] = 0, \quad i = 1, 2, \ldots, k.$$

We can represent the residual vector as follows,

$$\begin{aligned} r_k &= b - Ax_k = b - A(x_0 + Q_k f_k) = r_0 - \overline{Q_k} H_k f_k - b_k \overline{q_{k+1}} e_k^{\mathrm{T}} f_k \\ &= \overline{q_1}\|r_0\|_2 - \overline{Q_k} H_k f_k - b_k \overline{q_{k+1}} f_k^{(k)} \\ &= \overline{Q_1}(\|r_0\|_2 - H_k f_k) e_1 - b_k \overline{q_{k+1}} f_k^{(k)}, \end{aligned} \tag{2}$$

where $f_k^{(k)}$ is the k-th component of f_k. After doing inner product with $\overline{q_i}(i = 1, 2, \ldots, k)$ on both sides of the equation (2), we get

$$\|r_0\|_2 e_1 = H_k f_k. \tag{3}$$

Then the solution f_k is obtained by QR decomposition on H_k. More details can be found in the literature [5].

2.2. Implementation of CSYM

The CSYM algorithm can be described as follows.

Algorithm 1. The CSYM algorithm [3]

1) Choose initial vector $x_0 \in \mathbb{C}^n$, let $k := 0$, calculate

$$r_0 = b - Ax_0,\ q_0 = 0,\ q_1 = \frac{\overline{r_0}}{\|r_0\|_2},\ a_1 = [Aq_1, \overline{q_1}],\ c_{-1} = 0,\ b_0 = 0,\ s_{-1} = 0,$$
$$\tau_1 = \|r_0\|_2,\ c_0 = 0,\ s_0 = 0,\ p_{-1} = p_0 = 0\ .$$

2) The loop will continue until $\|r_k\|_2 < tol\|b\|_2$ is satisfied, where $tol \in \mathbb{R}_+$ (tolerance for termination); otherwise, calculate

$$\eta_k = c_{k-2}c_{k-1}b_{k-1} + \overline{s_{k-1}}a_k,\ \gamma_k = c_{k-1}a_k - c_{k-2}s_{k-1}b_{k-1},\ \theta_k = \overline{s_{k-2}}b_{k-1},$$
$$w = Aq_k - a_k\overline{q_k} - b_{k-1}\overline{q_{k-1}},\ b_k = \|w\|_2.$$

If $b_k = 0$, the loop stops; otherwise, calculate $q_{k+1} = \dfrac{\overline{w}}{b_k},\ a_{k+1} = [Aq_{k+1}, \overline{q_{k+1}}]$.

If $\gamma_k \neq 0,\ c_k = \dfrac{|\gamma_k|}{\sqrt{|\gamma_k|^2 + b_k^2}},\ s_k = \dfrac{\overline{\gamma_k}}{|\gamma_k|}\dfrac{b_k}{\sqrt{|\gamma_k|^2 + b_k^2}},\ \xi_k = \dfrac{\gamma_k}{|\gamma_k|}\sqrt{|\gamma_k|^2 + b_k^2}$;

otherwise, calculate $c_k = 0,\ s_k = 1,\ \xi_k = b_k,\ p_k = \dfrac{q_k - \eta_k p_{k-1} - \theta_k p_{k-2}}{\xi_k}$,

$$\tau_{k+1} = -s_k\tau_k,\ x_k = x_{k-1} + \tau_k c_k p_k,\ \|r_k\|_2 = |\tau_{k+1}|\ .$$

3) Let $k := k + 1$, and turn back to 2).

The following two propositions can be induced by the derivation of CSYM and algorithm 1.

Proposition 1. For the iteration x_k generated by the CSYM algorithm, there holds that $r_k = -b_k\overline{q_{k+1}}f_k^{(k)}$, where $f_k^{(k)}$ is the k-th component of f_k.

The analogous derivation of Proposition 1 can refer to the literature [3].

Proposition 2. For the residual vector r_k generated by the CSYM algorithm, there holds that

$$[r_k, r_j] = 0, \quad j = 1, 2, \ldots, k-1.$$

Proof. It is known that $q_1, q_2, \ldots, q_k$ are mutually orthogonal, i.e.,

$$[q_k, q_j] = 0, \quad j = 1, 2, \ldots, k-1,$$

hence,

$$[\overline{q_{k+1}}, \overline{q_{j+1}}] = q_{k+1}^{\mathrm{T}} \overline{q_{k+1}} = \overline{q_{k+1}^{\mathrm{H}} q_{k+1}} = \overline{[q_{k+1}, q_{j+1}]} = 0.$$

From the Proposition 1, the residual vector r_k satisfies

$$r_k = -b_k \overline{q_{k+1}} f_k^{(k)}, \tag{4}$$

there holds that

$$[r_k, r_j] = [-b_k \overline{q_{k+1}} f_k^{(k)}, -b_j \overline{q_{j+1}} f_j^{(j)}] = 0. \tag{5}$$

Remark 1. From above process, we observe that the premise of the calculation of a_{k+1} is to know the result of b_k, and they cannot be calculated synchronously. Therefore, we have to calculate twice inner products dependently for each b_k, a_{k+1}, which results in expensive calculations due to twice global communication between all processors.

In order to deal with this problem, three-term recurrence relation [16] and orthogonal properties of residual vectors is applied to replace the tridiagonalization process in this work during the calculation of the vector q_k and the tridiagonal matrix H_k with classical CSYM. The detailed implementation is given in the following section.

2.3. Improved CSYM algorithm

An improve CSYM algorithm (ICSYM) is proposed after adjusting the overall iteration steps, it will reduce the number of global communication per iteration. The ICSYM algorithm is described in detailed as follows.

According to the formula (1) and the formula (4), we have

$$b_k \overline{q_{k+1}} = A q_k - a_k q_k - b_{k-1} \overline{q_{k-1}},$$

and it is known that r_k is a constant of $\overline{q_{k+1}}$, thus the residual vector satisfies the following relationship,

$$r_{k+1} = \rho_k (r_k - \gamma_k A \overline{r_k}) + \mu_k r_{k-1}.$$

Choosing $A = O \in \mathbb{C}^{n \times n}$, the identical relation $r_j = b - A x_j \equiv b$ is always satisfied. Due to the arbitrariness of b, we can represent $b = \rho_k b + \mu_k b$ and $\mu_k = 1 - \rho_k$, then

$$r_{k+1} = \rho_k (r_k - \gamma_k A \overline{r_k}) + (1 - \rho_k) r_{k-1}. \tag{6}$$

According to the formula (6) the orthogonal properties of residual vectors, $[r_{k+1}, r_k] = 0$, $[r_{k-1}, r_k] = 0$, $[r_{k+1}, r_{k-1}] = 0$, we have

$$\gamma_k = \frac{[r_k, r_k]}{[A\overline{r_k}, r_k]},\ \rho_k = \frac{[r_{k-1}, r_{k-1}]}{[r_{k-1}, r_{k-1}] + \gamma_k [A\overline{r_k}, r_{k-1}]}.$$

Since

$$[A\overline{r_k}, r_{k-1}] = \overline{[r_k, A\overline{r_{k-1}}]} = [A\overline{r_{k-1}}, r_k],$$

$$A\overline{r_{k-1}} = -\rho_{k-1}^{-1}\gamma_{k-1}^{-1} r_k + \gamma_{k-1}^{-1} r_{k-1} + \rho_{k-1}^{-1}\gamma_{k-1}^{-1}(1-\rho_{k-1}) r_{k-2},$$

we can obtain

$$\rho_k = \left(1 - \frac{\gamma_k}{\gamma_{k-1}\rho_{k-1}} \frac{[r_k, r_k]}{[r_{k-1}, r_{k-1}]}\right)^{-1}.$$

Thus,

$$x_{k+1} = \rho_k (x_k + \gamma_k \overline{r_k}) + (1-\rho_k) x_{k-1}.$$

2.4. Implementation of ICSYM

The ICSYM algorithm based on the above manipulation can be given below.

Algorithm 2. The ICSYM algorithm

1) Choose initial vector $x_0 \in \mathbb{C}^n$, $\rho_0 = 1$, let $k := 0$, calculate

$$r_0 = b - Ax_0,\ \gamma_0 = \frac{[r_0, r_0]}{[A\overline{r_0}, r_0]},\ x_1 = \rho_0(x_0 + \gamma_0 \overline{r_0}),\ r_1 = \rho_0(r_0 - \gamma_0 A\overline{r_0}).$$

2) If $\|r_k\|_2 < tol\|b\|_2$ is satisfied, stop the loop, where $tol \in \mathbb{R}_+$; otherwise, calculate,

$$\gamma_k = \frac{[r_k, r_k]}{[A\overline{r_k}, r_k]},\ \rho_k = \left(1 - \frac{\gamma_k}{\gamma_{k-1}\rho_{k-1}} \frac{[r_k, r_k]}{[r_{k-1}, r_{k-1}]}\right)^{-1},$$

$$x_{k+1} = \rho_k (x_k + \gamma_k \overline{r_k}) + (1-\rho_k) x_{k-1},\ r_{k+1} = \rho_k (r_k - \gamma_k A\overline{r_k}) + (1-\rho_k) r_{k-1}.$$

3) Let $k := k + 1$, and turn back to 2).

Remark 2. From above process, we observed that two dependent inner products (a_{k+1} and b_k) are transformed into two independent inner products ($[r_k, r_k]$ and $[A\overline{r_k}, r_k]$), which only needs one global communication per iteration. Therefore, it can significantly reduce the global communication time on parallel distributed memory computers.

3. Theoretical analysis and parallel implementation

In order to indicate the parallel performance of both the CSYM and the ICSYM algorithms, we firstly mention a brief theoretical analysis of two algorithms, and then the corresponding parallel implementation will be discussed.

3.1. Theoretical analysis about two algorithms

The amount of calculation per iteration including several terms such as vector update (represents the additive operation between vectors), vector-multi (represents multiplication of vector), matrix-vector (represents the matrix and vector multiplication), inner product and All-reduce (represents all-reduce operator) are given in Table 1.

Table 1
The amount of calculation per iteration

Algorithm	Vector update	Vector-multi	Matrix-vector	Inner product	All-reduce
CSYM	5	7	1	2	2
ICSYM	4	6	1	2	1

From Table 1, we can see that the ICSYM algorithm needs only four vector updates and six vector multiplications which are less than the number needed (5 and 7) in the CSYM algorithm. As we know the dominant time depends on vector multiplications and global communication at each iteration, the proposed ICSYM can decrease the reduce-operator per iteration from two to one communication. Therefore, the ICSYM algorithm has better parallel performance than CSYM.

3.2. Parallel implementation of ICSYM

In this section we discuss the parallel implementation of the ICSYM algorithm including data storage and implementation of every iteration.

3.2.1. Data storage

For convenience let p be the number of processors, p_i $(i = 1, 2, \ldots, p)$ represent ith processor and l is integer in $n = pl$.

Mark

$$A = (A_1^{\mathrm{T}}, A_2^{\mathrm{T}}, \cdots, A_p^{\mathrm{T}})^{\mathrm{T}},\ b = (b_1^{\mathrm{T}}, b_2^{\mathrm{T}}, \cdots, b_p^{\mathrm{T}})^{\mathrm{T}},\ s = (s_1^{\mathrm{T}}, s_2^{\mathrm{T}}, \cdots, s_p^{\mathrm{T}})^{\mathrm{T}},$$
$$x^{(k)} = ((x_1^{(k)})^{\mathrm{T}}, (x_2^{(k)})^{\mathrm{T}}, \cdots, (x_p^{(k)})^{\mathrm{T}})^{\mathrm{T}},\ r^{(k)} = ((r_1^{(k)})^{\mathrm{T}}, (r_2^{(k)})^{\mathrm{T}}, \cdots, (r_p^{(k)})^{\mathrm{T}})^{\mathrm{T}},$$

where A_i is a $l \times n$ sub-block matrix, b_i, $x_i^{(k)}$, $r_i^{(k)}$, s_i are $l \times 1$ vectors $(k = 0, 1, 2, \ldots)$, which are stored on the processor p_i $(i = 1, 2, \cdots, p)$. The similar storage manners can be found in the reference [4]. Detailed descriptions of parallel computing is mentioned in the following.

3.2.2. Parallel implementation of ICSYM

In each processor $p_i (i = 1, 2, \cdots, p)$, computing process and cycling process have to be implemented. Moreover, the most important point of the parallel implementation is all-reduce operator involved in each processor p_i simultaneously, which collects values

from all processors and distributes the result back to all processors. In this work, the sum operator is applied in the all-reduce.

1) Computing process:

Given $x^{(0)}$ and $\rho_0 = 1$, calculate

$$r_i^{(0)} = b_i - A_i x^{(0)},\ s_i = A_i \overline{r^{(0)}},\ [r_i^{(0)}, r_i^{(0)}] \text{ and } [s_i, r_i^{(0)}],$$

after one all-reduce, we can obtain $[r^{(0)}, r^{(0)}]$ and $[s, r^{(0)}]$, then calculate

$$\gamma_0 = \frac{[r^{(0)}, r^{(0)}]}{[s, r^{(0)}]},\ x_i^{(1)} = x_i^{(0)} + \gamma_0 \overline{r_i^{(0)}},\ r_i^{(1)} = r_i^{(0)} - \gamma_0 s_i.$$

2) Cycling process:

For $k = 1, 2, \cdots$, calculate

$$s_i = A_i \overline{r^{(k)}},\ [r_i^{(k)}, r_i^{(k)}] \text{ and } [s_i, r_i^{(k)}],$$

we can get $[r^{(k)}, r^{(k)}]$ and $[s, r^{(k)}]$ after all-reduce, then calculate

$$\gamma_k = \frac{[r^{(k)}, r^{(k)}]}{[s, r^{(k)}]},\ \rho_k = \left(1 - \frac{\gamma_k}{\gamma_{k-1}\rho_{k-1}} \frac{[r^{(k)}, r^{(k)}]}{[r^{(k-1)}, r^{(k-1)}]}\right)^{-1},$$

$$x_i^{(k+1)} = \rho_k(x_i^{(k)} + \gamma_k \overline{r_i^{(k)}}) + (1 - \rho_k) x_i^{(k-1)},\ r_i^{(k+1)} = \rho_k(r_i^{(k)} - \gamma_k s_i) + (1 - \rho_k) r_i^{(k-1)}.$$

Remark 3. As we can see from the above parallel implementation, only one time global communication (all-reduce) at each iteration is needed to calculate for $[r^{(k)}, r^{(k)}]$ and $[s, r^{(k)}]$.

3.3. Parallel analysis of CSYM and ICSYM

In order to further illustrate the parallel performance, we give a detailed parallel analysis which is similar to the literature [21].

According to the literature [21], we can know that the time of a vector update without communication or the time of a vector-multi is

$$t_{vec-upd} = t_{vec-mul} = 2t_{fl} n/p,$$

where t_{fl} denotes the time for a floating point operation.

The time of a matrix-vector is

$$t_{mat-vec} = (2n_z - 1)t_{fl} n/p + 2n_m t_s + 2(2n_b + n_m)t_w,$$

and the time of k inner products which only need one parallel computation is

$$t_{inn-prod}(k) = 2kt_{fl} n/p + 2(t_s + kt_w) logp,$$

where n_z is the number of nonzero elements at each row, n_m represents the number of messages sent and received, n_b is the number of boundary data elements per processor, t_s is the start time of communication, t_w is the time required to transmit a word between two processors.

Therefore, we can observe that the time of two algorithms at each iteration is

$$\begin{aligned} T_{CSYM} &= 5t_{vec-upd} + 7t_{vec-mul} + t_{mat-vec} + 2t_{inn-prod}(1) \\ &= (2n_z + 27)t_{fl}n/p + 4(t_s + t_w)logp + 2n_m t_s + 2(2n_b + n_m)t_w, \end{aligned}$$

and

$$\begin{aligned} T_{ICSYM} &= 4t_{vec-upd} + 6t_{vec-mul} + t_{mat-vec} + t_{inn-prod}(2) \\ &= (2n_z + 23)t_{fl}n/p + 2(t_s + 2t_w)logp + 2n_m t_s + 2(2n_b + n_m)t_w, \end{aligned}$$

respectively.

Because $t_f < t_w \ll t_s$ is satisfied, and n_m can be ignored in the distributed parallel computer, we can obtain $T_{ICSYM} < T_{CSYM}$. Moreover, the number of processors for minimum parallel time about two algorithms can be obtained by taking the partial derivatives, i.e.,

$$p_{CSYM} = \frac{(2n_z + 27)t_{fl}nln2}{4(t_s + t_w)},$$

and

$$p_{ICSYM} = \frac{(2n_z + 23)t_{fl}nln2}{2(t_s + 2t_w)}.$$

Then,

$$\frac{p_{ICSYM}}{p_{CSYM}} \approx 2.$$

Hence, ICSYM has the better scalability than CSYM.

In addition, when n is fixed and p is large enough, we can get that the performance improving rate of ICSYM is

$$\eta = \frac{T_{CSYM}}{T_{ICSYM}} \approx \frac{4t_{fl}n + 2t_s plogp}{(2n_z + 27)t_{fl}n + 4t_s plogp} \to 50\%.$$

The data further illustrates that the improving rate of ICSYM compared with ICSYM can reach 50% in theroy.

Finally, we provide the scalability analysis about two algorithms.

According to the definition of iso-efficiency function [11, 21], $T^{se} = \frac{E}{1-E}T^{cost}$, and $T^{cost} = pT^{par} - T^{se}$, where T^{par} denotes the parallel time, T^{se} denotes the sequential time, T^{cost} presents the extra time, E is the parallel efficiency.

Hence, we can obtain the correlation formula between the number of processors and the parallel efficiency of two algorithms,

$$n_{CSYM} = \frac{4(t_s + t_w)E}{(2n_z + 27)t_{fl}(1 - E)} plogp \approx \frac{4t_s E}{(2n_z + 27)t_{fl}(1 - E)} plogp,$$

$$n_{ICSYM} = \frac{2(t_s + 2t_w)E}{(2n_z + 23)t_{fl}(1 - E)} plogp \approx \frac{2t_s E}{(2n_z + 23)t_{fl}(1 - E)} plogp.$$

From above analysis, in the case of equal parallel computational efficiency, the number of processors of CSYM is about twice times than that of ICSYM when p increases. Therefore, the parallelism and scalability of ICSYM is better than that of CSYM.

4. Numerical experiments

We provide different numerical experiments to illustrate the performance of the ICSYM algorithm in this section. Then the detailed numerical results are depicted.

Test case I

In order to evaluate the performance of ICSYM, we first consider two practical problems arising from quantum chemistry [1], quantum chemistry (QC2534) and quantum chemistry (QC324) respectively. Then we report a random example to further verify the applicability of ICSYM for dense linear system, the coefficient matrix consisting of a great number of nonzero elements is generated via the xLATMR routine in LAPACK. Additionally, the right-hand side b for each test problem is chosen as $(1 + i, \ldots, 1 + i)^{\mathbb{T}}$. The number of nonzero elements for three matrices is shown in Table 2. Numerical results compared with five algorithms (ICSYM, CSYM, COCR [7, 13], COCG [7] and QMR-SYM [14]) can be seen in Table 3. Convergence histories of the different iterative algorithms for each test problem are illustrated from Figure 1 to Figure 3.

These cases have been carried out in MATLAB R2017b with a Windows 7 (64 bit) on Inter(R) Core(TM) i7-6700 CPU 3.40Ghz and 16.00GB of RAM. Then we compare with five algorithms (ICSYM, CSYM, COCR, COCG and QMR-SYM) in terms of number of iterations (abbreviated as Iters), CPU consuming time in seconds (abbreviated as CPU). The case in terms of Iters, CPU is reported by means of tables while convergence histories are shown in figure with Iters (on the horizontal axis) versus $\log_{10}$ of the updated relative residual 2-norm which defines as $\log_{10} \frac{\|b - Ax_k\|_2}{\|b\|_2}$ (on the vertical axis).

In our implementations, the initial guess is chosen to be zero vector and the stopping criteria for all algorithms is $tol = 10^{-8}$, where x_k is the current approximation. In addition, Size stands for the number of rows (M) and columns (N) in the matrix, expressed as $M \times N$. Num-nonzero represents the number of nonzero elements in

a matrix. Density means the percentage of nonzero elements. The "–" which appears in Table 3 means prohibitive CPU time and the number of iterations.

Table 2
The number of nonzero elements for three matrices

Matrix	QC2534	QC324	RANDOM MATRIX
Size	2534×2534	324×324	800×800
Num-nonzero	463 360	26 730	599 800
Density	**7.22%**	**25.46%**	**93.72%**

Table 3
Numerical results of five algorithms for three matrices

Matrix	QC2534		QC324		RANDOM MATRIX	
	Iters	CPU	Iters	CPU	Iters	CPU
ICSYM	**1047**	**1.94**	**847**	**0.07**	**1630**	**0.90**
CSYM	**1066**	**1.95**	**855**	**0.09**	**1639**	**1.07**
COCR	5003	4.41	1625	0.10	–	–
COCG	5067	4.32	1598	0.09	–	–
QMR-SYM	–	–	2830	0.30	–	–

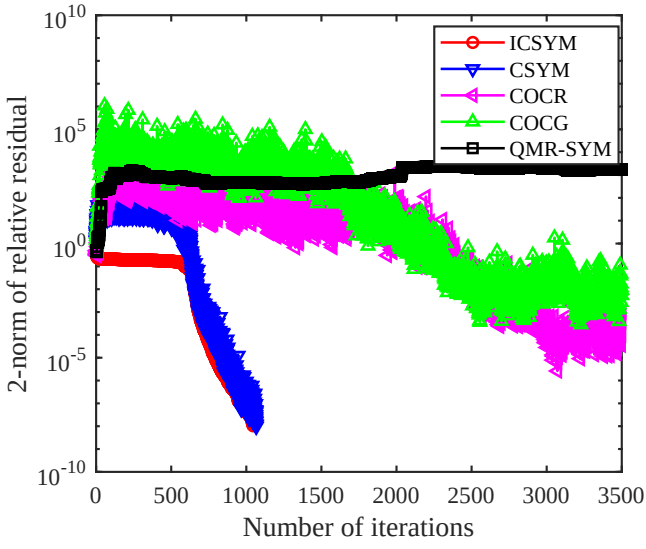

Figure 1. Convergence histories for QC2534 with the sparse coefficient matrix

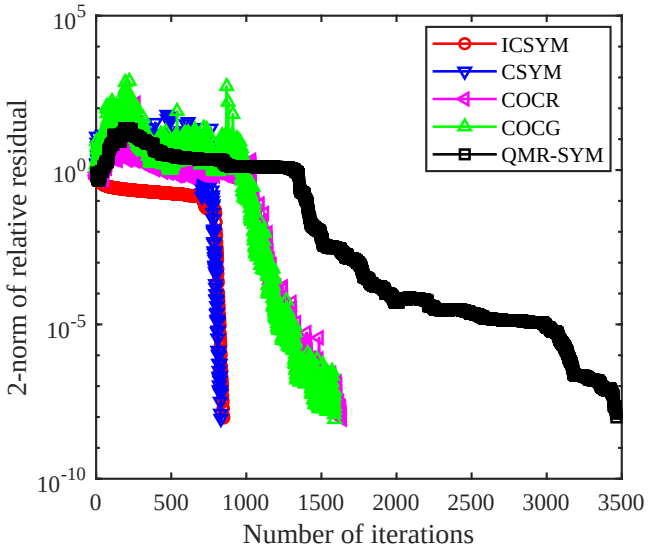

Figure 2. Convergence histories for QC324 with the relatively dense coefficient matrix

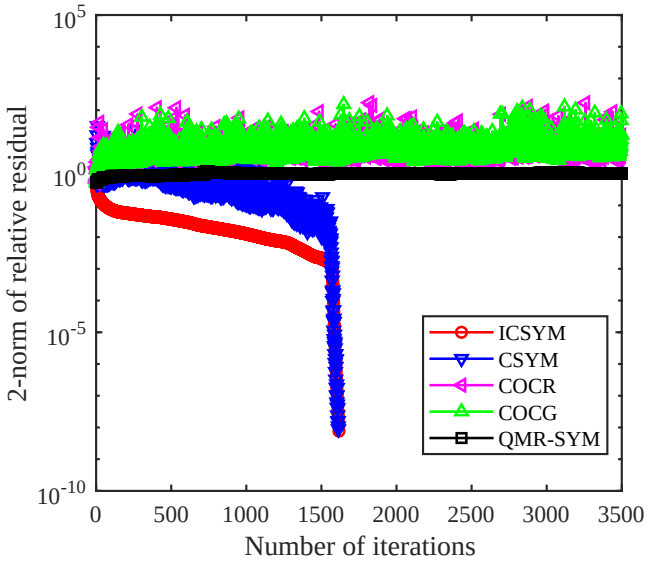

Figure 3. Convergence histories for RANDOM MATRIX with the dense coefficient matrix

According to the characteristics of three matrices in Table 2, matrix QC2534 is a sparse complex symmetric matrix, and matrix QC324 is a relatively dense complex symmetric matrix. On the contrary, the random matrix with more nonzero elements

given in the fourth column of Table 2 is a dense complex symmetric matrix. From the numerical results shown in Table 3, we can observe that both the ICSYM and the CSYM algorithms are feasible and effective for solving all types of complex symmetric linear systems. Additionally, we can see clearly that the number of iterations and CPU time of both the ICSYM and the CSYM algorithms are less than that of other three methods (COCR, COCG and QMR-SYM) for solving all types of complex symmetric linear systems. Moreover, it can be seen that the ICSYM algorithm requires less CPU time than that of the CSYM algorithm. This is normal because the amount of vector updates and vector multiplications per iteration step in ICSYM is less than that in CSYM.

The phenomenon of convergence rates can be further exhibited from Figure 1 to Figure 3, it is much faster in CSYM and ICSYM than that of other three methods (COCR, COCG and QMR-SYM). From Figure 3, we observe that three algorithms (COCR, COCG and QMR-SYM) fail to converge and have not a downward trend even using 3500 iteration steps, it is further illustrated that both the ICSYM and the CSYM algorithms are the best solvers for solving dense complex symmetric linear systems among the five algorithms. In addition, there are shown the same phenomena from Figure 1 to Figure 3 that the convergence curves of both the ICSYM and the CSYM algorithms are almost coincident when reaches a certain number of iterations. That means that the indeed idea of ICSYM is the same as CSYM.

In conclusion, both the ICSYM and the CSYM algorithms have almost the same numerical stability and are recommended to solve all types of complex symmetric linear systems arising from quantum chemistry. Furthermore, it is more efficient for solving relatively dense or dense complex symmetric linear systems. Therefore, in order to verify the parallel performance of the improved algorithm, the following case is discussed in detailed for solving large dense linear systems and compared with classical CSYM.

Test case II

In this case, the results evaluations of parallel performance about two algorithms (ICSYM and CSYM)are given in Figure 4 and Figure 5. The dimension of matrix equals 40000, and the density (the proportion of non-zero elements) is more than 70%. The case has been carried out in the parallel machine Lenovo Shenteng 1800 cluster.

Note: The speedup shown in Figure 4 is computed as the ratio of the parallel computing time and that using one processor. Saving computational time is obtained by $1 - T_A(p)/T_B(p)$ and depicted in Figure 5, where $T_A(p)$ and $T_B(p)$ are the computing times on p processors of ICSYM and CSYM respectively.

From the results in Figure 4, we can observe distinctly that the speed up of the ICSYM algorithm is faster than that of the CSYM algorithm. Moreover the speedup has a remarkable growth with increasing number of processors compared with the CSYM algorithm. From Figure 5, the ratio of the saving computational time of ICSYM becomes larger with increasing the number of processors.

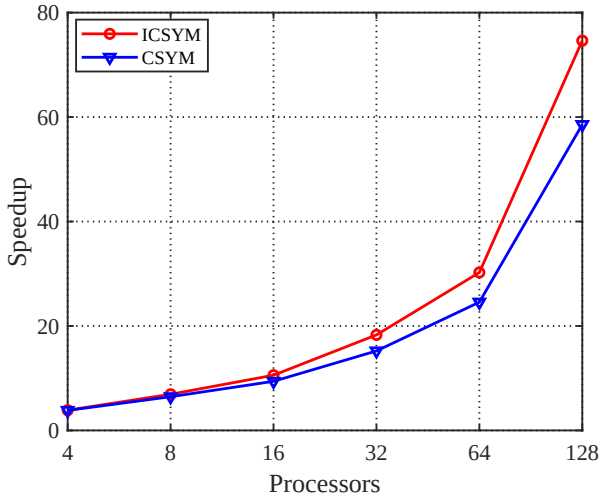

Figure 4. Experimental results of speed up

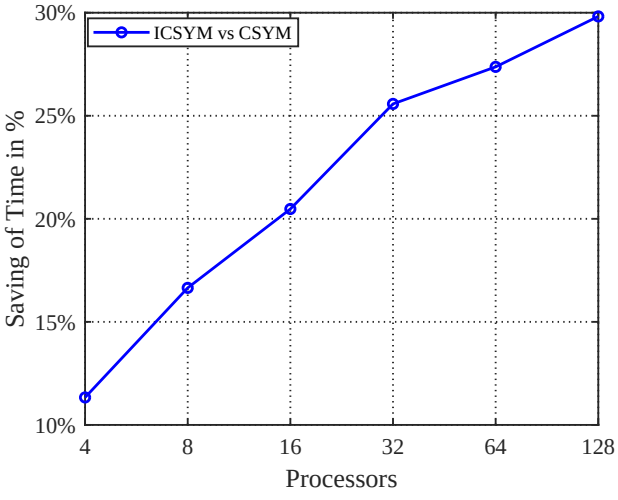

Figure 5. Execution time reduction

5. Conclusions

In this paper, a parallel algorithm called ICSYM is proposed for solving large-scale complex symmetric linear systems. The numerical test results are shown that:

(1) ICSYM algorithm is available for all types of complex symmetric linear systems. In the comparison with other four algorithms mentioned (CSYM, COCR,

COCG and QMR-SYM), it is shown that the ICSYM algorithm is more efficient for solving complex symmetric linear systems. Moreover, it is more obvious for solving large relatively dense or dense complex symmetric linear systems.

(2) For the large-scale complex symmetric linear systems, the ICSYM algorithm not only has better parallelism than the CSYM algorithm but also has the similar numerical stability.

In conclusion, the ICSYM algorithm is high-efficiency and good stable for solving complex symmetric linear systems, particularly large-scale dense linear systems.

Future research will address to develop preconditioning strategies to enhance the performance, particularly in the large complex symmetric linear systems from the practical engineering applications.

Acknowledgements

This study was funded by the National Natural Science Foundation of China (Grants No. 11302173 and No. 11620101002) and the Natural Science Foundation of Shaanxi Province (Grant No. 2017JQ1037). The computation of case II was executed in the Center for High Performance Computing of Northwestern Polytechnical University.

References

[1] Bai Z., Day D., Demmel J., Dongarra J.: A Test Matrix Collection for Non--Hermitian Eigenvalue Problems. In: *Technical Report CS-97-355*, 1997.

[2] Bücker H.M., Sauren M.: A parallel version of the Quasi-Minimal Residual method based on coupled two-term recurrences, *International Workshop on Applied Parallel Computing*, vol. 1184, pp. 157–165, 1996.

[3] Bunse-Gerstner A., Stöver R.: On a conjugate gradient-type method for solving complex symmetric linear systems, *Linear Algebra and its Applications*, vol. 287(1–3), pp. 105–123, 1999.

[4] Chen G., Hong A., Chen J., Zheng Q., Shan J.L.: *The Parallel Algorithm.* Higher Education Press, Beijing, 2004.

[5] Clemens M., Weiland T., Van Rienen U.: Comparison of Krylov-type methods for complex linear systems applied to high-voltage problems, *IEEE Transactions on Magnetics*, vol. 34, pp. 3335–3338, 1998.

[6] Elman H.C., O'Leary D.P.: Eigenanalysis of some preconditioned Helmholtz problems, *Numerische Mathematik*, vol. 83(2), pp. 231–257, 1999.

[7] Gu X., Clemens M., Huang T., Li L.: The SCBiCG class of algorithms for complex symmetric linear systems with applications in several electromagnetic model problems, *Computer Physics Communication*, vol. 191, pp. 52–64, 2015.

[8] Hu Q., Yuan L.: A Plane-Wave Least-Squares Method for Time-Harmonic Maxwell's Equations in Absorbing Media, *SIAM Journal on Scientific Computing*, vol. 36(4), pp. A1937–A1959, 2014.

[9] Hu Q., Yuan L.: A Weighted Variational Formulation Based on Plane Wave Basis for Discretization of Helmholtz Equations, *International Journal of Numerical Analysis and Modeling*, vol. 11(3), pp. 587–607, 2014.

[10] Huttunen T., Malinen M., Monk P.: Solving Maxwell's equations using the ultra weak variational formulation, *Journal of Computational Physics*, vol. 223(2), pp. 731–758, 2007.

[11] Kumar V., Rao V.N.: Parallel depth first search. Part II. Analysis, *International Journal of Parallel Programming*, vol. 18, pp. 501–509, 1987.

[12] Li C., Qiao Z.: A Fast Preconditioned Iterative Algorithm for the Electromagnetic Scattering From a Large Cavity, *Journal of Scientific Computing*, vol. 53, pp. 435–450, 2012.

[13] Sogabe T., Zhang S.-L.: A COCR method for solving complex symmetric linear systems, *Journal of Computational and Applied Mathematics*, vol. 199, pp. 297–303, 2007.

[14] Van der Vorst H.: *Iterative Krylov Methods for Large Linear Systems*. Cambridge University Press, Cambridge, 2003.

[15] Van der Vorst H.A., Melissen J.B.M.: A Petrov-Galerkin type method for solving $Ax = b$, where A is symmetric complex, *IEEE Transactions on Magnetics*, vol. 26, pp. 706–708, 1990.

[16] Wu J., Wang Z., Li X.: *Efficient solution and parallel computation of sparse linear equations*. Hunan Science and Technology Press, Changsha, 2004.

[17] Xiang T., Liang C.: Iterative solution for dense linear systems arising in computation electromagnetics, *Journal of Xidian University*, vol. 30, pp. 748–751, 2003.

[18] Yang L.T., Brent R.P: The improved BiCG method for large and sparse unsymmetric linear systems on parallel distributed memory architectures, *International Parallel and Distributed Processing Symposium*, vol. 3, pp. 1–7, 2002.

[19] Yang L.T., Brent R.P.: The improved BiCGStab method for large and sparse nonsymmetric linear systems on parallel distributed memory architectures. In: *Proceedings of Fifth International Conference on Algorithms and Architectures for Parallel Processing*, pp. 324–328, 2002.

[20] Zhag L.-T., Zuo X.-Y., Gu T.-X., Huang T.-Z., Yue J.-H.: Conjugate residual squared method and its improvement for non-symmetric linear systems, *International Journal of Computer Mathematics*, vol. 87, pp. 1578–1590, 2010.

[21] Zuo X., Liu Y., Zhang L., Meng H.: A parallel version of COCR method for solving complex symmetric linear systems, *Parallel and Cloud Computing Research*, vol. 2(1), pp. 12–18, 2014.

Affiliations

Yingchun Zhang
School of Mechanical Engineering, Northwestern Polytechnical University, Box 552, Xi'an, 710072, Shaanxi, China, zhangyingchun@mail.nwpu.edu.cn,
ORCID ID: https://orcid.org/0000-0003-2499-4444

Quanyi Lv
Department of Applied Mathematics, Northwestern Polytechnical University, Xi'an 710072, Shaanxi, China, luquan@nwpu.edu.cn

Manyu Xiao
Department of Applied Mathematics, Northwestern Polytechnical University, Xi'an 710072, Shaanxi, China, manyuxiao@nwpu.edu.cn,
ORCID ID: https://orcid.org/0000-0001-8121-290X

Gongnan Xie
School of Marine Science and Technology, Northwestern Polytechnical University, Box 24, Xi'an, 710072, Shaanxi, China, xgn@nwpu.edu.cn,
ORCID ID: https://orcid.org/0000-0003-1047-4990

Piotr Breitkopf
Sorbonne Universités, Université de Technologie de Compiègne, Laboratoire Roberval, France, Piotr.breitkopf@utc.fr, ORCID ID: https://orcid.org/0000-0002-3842-6520

Received: 25.04.2018
Revised: 03.08.2018
Accepted: 08.08.2018