A Parallel Version of COCR Method for Solving Complex Symmetric Linear Systems

Xianyu Zuo*1, Yang Liu1, Litao Zhang2, He Meng1

1 Institute of Data and Knowledge Engineering, Henan University, Kaifeng 475000, P.R. China

2 Zhengzhou Institute of Aeronautical Industry Management, Zhengzhou, Henan, 450015, P. R. China

*1 xianyu_zuo@163.com

Received 14 October 2013; Accepted 6 January 2014; Published 20 January 2014

Abstract

In this paper, a parallel Conjugate Orthogonal Conjugate Residual (COCR) method (PCOCR method, in brief) for solving complex symmetric linear systems is proposed for distributed parallel environments. The method reduces two global synchronization points to one by reconstructing COCR method and all inner products per iteration are independent and communication time required for inner product can be overlapped efficiently with computation time of vector updates. It combines the elements of numerical stability with the characters of designed parallel algorithms. The cost is only slightly increased computation. Performance analysis shows that PCOCR method has better parallelism and scalability than COCR method and the global communication performance can be improved 50%. Meanwhile, numerical experiments show that PCOCR and COCR have the same numerical stability.

Keywords

PCOCR Method; Krylov Subspace; Complex Symmetric Linear Systems; Synchronization Overhead; W Parallel Computing

Introduction

One of the fundamental tasks of numerical computation is to solve linear systems. These systems arise very frequently in scientific computing, for example, from finite difference or finite element discretization of partial differential equations, as intermediate steps in finding the solution of nonlinear problems or as sub-problems in linear and nonlinear programming. Usually, these systems are large and sparse and solved by iterative methods. Among the iterative methods for large sparse systems, Krylov subspace methods are the most powerful; for example, conjugate gradient (CG) method for solving symmetric positive definite linear systems, the GMRES method, BiCG method, QMR method, BiCGStab method and BiCR method for solving unsymmetrical linear systems, the COCG method, the COCR method for solving complex symmetric linear systems and so on. However, the Krylov subspace methods suffer bottleneck, i.e. the global communication, when used in large scale parallel computing.

The basic time-consuming computational kernels of all Krylov subspace methods, involving COCR method, are usually: inner products, vector updates and matrix-vector multiplications. In many situations, especially when matrix operations are well structured, these operations are suitable for implementation on vector and shared memory parallel computers. But for parallel distributed memory machines, the matrices and vectors are distributed over the processors, so that even when the matrix operations can be implemented efficiently by parallel operations, we still can't avoid the global communication, i.e. accumulation of data from all to one processor and broadcast the results to all processors required for inner product computations. Vector updates are naturally parallel and, for large sparse matrices, matrix-vector multiplications can be implemented with communication between only nearby processors. The bottleneck is usually due to inner products enforcing global communication whose costs become relatively more important when the number of parallel processors is increased and thus they have the potential to affect the scalability of the algorithm in a negative way. The detailed discussion on the communication problem on distributed memory systems can be found in.

Three remedies can be used to solve the bottleneck leading to performance degeneration. The first is to eliminate data dependency, and several resulting

inner products can be computed and passed at the same time. The second is to reconstruct algorithm, resulting communication and computation can be overlapped efficiently. The last is to replace computation involving global communications by the other computation without global communications. Of course, the three strategies can be used connectedly. The remedy, used in this paper, belongs to the first, i.e. reducing the global communication times or number of global synchronization points.

Biicker et al. and Yang et al. proposed a new parallel Quasi-Minimal Residual (QMR) method based on coupled two-term recurrences Lanczos process. Sturler et al. gave suggestion on how to reduce affection of the global communication in GMRES(m) and CG methods. Yang et al. proposed the improved CGS, BiCG and BiCGStab methods respectively. Liu et al. gave an improved CR algorithm. Gu et al. proposed the improved BiCR and GPBiCG methods. Collignon and Van Gijzen proposed the parallel variants of IDR(s) method. All of these methods depended on the former two strategies. Gu, Liu and Mo presented a CG-type method without global inner products, i.e. multiple search direction conjugate gradient (MSD-CG) method. Based on domain discretion, MSD-CG method replaced the inner products computation in CG method by small size linear systems. Therefore, it eliminates global inner products completely, which belongs to the last remedy.

Based on the former two remedies, a parallel COCR method is put forward. The algorithm is reorganized without changing the numerical stability so that all inner products of a single iteration step are independent (only one single global synchronization point), and subsequently communication time required for inner products can be overlapped efficiently with computation time. The cost is only a little increased computation. Performance analysis shows that PCOCR method has better parallelism and scalability than COCR method. Especially, the parallel performance can be improved by a factor of about 2.

The paper is organized as follows. Section 2 presents the algorithm design of PCOCR method. Performance analysis about the two methods is given in section 3. In section 4, iso-efficiency analysis about the two methods is presented. Numerical experiments are reported in section 5. Finally conclusion is made in section 6.

Algorithm Design of PCOCR Method

Consider solving a large sparse complex symmetric linear system

$$Ax = b \tag{1}$$

on a parallel distributed memory machine, where $A \in C^{N\times N}$, $x, b \in C^N$.

For comparison, we give COCR method for (1) discussed in, where x_0 and $r_0 = b - Ax_0$ be the initial guess and residual vector, respectively, such that $r_0^T r_0 \neq 0$.

Algorithm 1. (COCR Method, Sogabe,2007)

1) *Compute* $r_0 = b - Ax_0$, $p_{-1} = 0$, $\beta_{-1} = 0$;
2) ***For*** *n=0,1,2,3...,until convergence,* ***do***
3) $p_n = r_n + \beta_{n-1} p_{n-1}$;
4) $Ap_n = Ar_n + \beta_{n-1} Ap_{n-1}$;
5) $\alpha_n = (\bar{r}_n, Ar_n) / (\overline{Ap}_n, Ap_n)$;
6) $x_{n+1} = x_n + \alpha_n p_n$;
7) $r_{n+1} = r_n - \alpha_n Ap_n$;
8) $\beta_n = (\bar{r}_{n+1}, Ar_{n+1}) / (\bar{r}_n, Ar_n)$;
9) ***End do***

In Algorithm 1, steps 5) and step 8) require inner products, and other computing steps are closely connected with these inner products. Furthermore, steps 5) and step 8) have close data dependency. So there are two global synchronization points per iteration. These global communication costs become relatively more and more important when the number of parallel processors is increased and thus they have the potential to affect the scalability of the algorithm in a negative way.

The main idea of PCOCR algorithm is to eliminate data dependency of inner products computing between steps 5) and step 8) through mathematical derivation.

It is defined that $q_n = Ap_n$, $\bar{q}_n = \overline{Ap}_n$, step 5) in COCR method becomes

$$\alpha_n = (\bar{r}_n, Ar_n) / (\overline{Ap}_n, Ap_n) = (\bar{r}_n, Ar_n) / (\bar{q}_n, q_n) \tag{2}$$

step 4) becomes

$$q_n = Ar_n + \beta_{n-1} q_{n-1} \tag{3}$$

from equation (3), we can get

$$\begin{aligned}(\bar{q}_n, q_n) &= (\overline{Ar}_n + \beta_{n-1}\bar{q}_{n-1}, Ar_n + \beta_{n-1} q_{n-1}) \\ &= (\overline{Ar}_n, Ar_n) + 2\beta_{n-1}(\overline{Ar}_n, q_{n-1}) \\ &\quad + \beta^2_{n-1}(\bar{q}_{n-1}, q_{n-1})\end{aligned} \tag{4}$$

Define that $\rho_n = (\bar{r}_n, Ar_n)$, $\delta_n = (\bar{q}_n, q_n)$, $\varsigma_n = (\overline{Ar}_n, Ar_n)$ and $\eta_n = (\overline{Ar}_n, q_{n-1})$, step 8) becomes

$$\beta_n = (\bar{r}_{n+1}, Ar_{n+1}) / (\bar{r}_n, Ar_n) = \rho_{n+1} / \rho_n \tag{5}$$

The parallel COCR method can be presented in algorithm form as follows:

Algorithm 2. (PCOCR Method)

1) *Compute* $r_0 = b - Ax_0$, $p_{-1} = q_{-1} = 0$, $\beta_{-1} = 0$, $\rho_0 = (\bar{r}_0, Ar_0)$, $\varsigma_0 = (\overline{Ar}_0, Ar_0)$;

2) ***For*** *n=0,1,2,3...,until convergence,* ***do***

3) $p_n = r_n + \beta_{n-1} q_{n-1}$;

4) $q_n = Ar_n + \beta_{n-1} q_{n-1}$;

5) $\delta_n = \varsigma_n + 2\beta_{n-1}\eta_n + \beta_{n-1}^2 \delta_{n-1}$;

6) $\alpha_n = \rho_n / \delta_n$;

7) $x_{n+1} = x_n + \alpha_n p_n$;

8) $r_{n+1} = r_n - \alpha_n q_n$;

9) $\rho_{n+1} = (\bar{r}_{n+1}, Ar_{n+1})$, $\varsigma_{n+1} = (\overline{Ar}_{n+1}, Ar_{n+1})$, $\eta_{n+1} = (\overline{Ar}_{n+1}, q_n)$;

10) $\beta_n = \rho_{n+1} / \rho_n$

11) ***End do***

So we have reduced global synchronization points from two (Algorithm 1) to one (Algorithm 2) and kept the number of vector updates and matrix-vector multiplications constant without changing the computation order of the original method. As the inner products in step 9) are independent with each other, the new PCOCR method need only one global synchronization point in each iterative step.

Performance Analysis of Both Methods

The derivation of PCOCR method shows that PCOCR and COCR methods as well as COCR methods are mathematically equivalent. The amount of calculation per iteration for each method without preconditioning is given in Table 1.

TABLE 1 THE AMOUNT OF CALCULATION PER ITERATION

Methods	Vector update	Matrix-Vector	Inner product	Global Synchro.
COCR	4	1	2	2
PCOCR	4	1	3	1

From TABLE 1, it can be seen that PCOCR method needs four vector updates update and three inner productions more than COCR method. But the global synchronization points per iteration have been reduced from two to one. The increased amount of computation is slightly relative to the reduction of global communication cost.

In the following discussion, algorithm analysis is based on distributed memory parallel machine which is a mesh-based processor grid with P processors. Each processor has its own memory and operation units. All of operation units execute the same program, i.e. Single-Program and Multi -Data (SPMD) model. If one of processor needs data from other processor, they are transformed by message passing and communication is carried out through binary tree way.

A performance analysis of both methods on distributed memory parallel computers has been made. Some similar denotations are as follows. P is the number of processors. N is the total number of unknowns. n_z is the average number of nonzero elements per row in matrix A. t_{fl} is the average time for a double precision floating point operation. t_s denotes the communication start up time. t_w is the transmission time of a word between two neighboring processors.

Since the computational and communication patterns are the same per iteration, only time complexity of the parallel computation and communication of one iteration step is taken into account.

For a vector update (daxpy) or an inner product (ddot), the computation time is given as $2t_{fl}N/P$, where N/P is the local number of unknowns on a processor. The computation time for the (sparse) matrix-vector product is given as $(2n_z - 1)t_{fl}N/P$.

The global accumulation and broadcast time for one inner product is taken as $2\log P(t_s + t_w)$, while the global accumulation and broadcast for k simultaneous inner products as $2\log P(t_s + kt_w)$. Assume that coefficient matrix is mapped to processors such that the matrix-vector needs a processor that only communicates with the nearest neighbor processors. The communication for the matrix-vector product is necessary for the exchange of so-called boundary data: sending boundary data to other processors and receiving boundary data from other processors. Assume that each processor has to send and receive n_m messages, and let the number of boundary data elements on a processor be given by n_b. The total number of words that have to be communicated (sent and received) is then $2(2n_b + n_m)$ per processor. For both methods, the communication time of one matrix-vector product is $2n_m t_s + 2(2n_b + n_m)t_w$.

In summary, the time of a vector update is, since it needs no communication, that

$$t_{vec\_upd} = 2t_{fl}N / P \quad (6)$$

The time for k simultaneous inner products is

$$t_{inn\_prod}(k) = 2kt_{fl}N / P + 2\log P(t_s + kt_w) \quad (7)$$

and the time for a matrix-vector is

$$t_{mat\_vec} = (2n_z - 1)t_{fl}N / P + 2n_m t_s + 2(2n_b + n_m)t_w \quad (8)$$

From Table 1, the time per iteration of COCR method is

$$\begin{aligned} T_{COCR} &= 4t_{vec\_upd} + 2t_{inn\_prod}(1) + t_{mat\_vec} \\ &= (2n_z + 11)t_{fl}N / P + 4\log P(t_s + t_w) \\ &\quad + 2n_m t_s + 2(2n_b + n_m)t_w \end{aligned} \quad (9)$$

and of PCOCR is

$$\begin{aligned} T_{PCOCR} &= 4t_{vec\_upd} + t_{inn\_prod}(3) + t_{mat\_vec} \\ &= (2n_z + 12)t_{fl}N / P + 2\log P(t_s + 3t_w) \\ &\quad + 2n_m t_s + 2(2n_b + n_m)t_w \end{aligned} \quad (10)$$

It is known that $t_s \gg t_w$ for massively distributed parallel computer. By comparison between equation (9) and (10), it is obtained that the parallelism of PCOCR method is better than that of COCR method since $T_{PCOCR} < T_{COCR}$.

Minimizing T_{COCR} and T_{PCOCR} form (9) and (10), it is inferred that the number of processors for minimal parallel time of both methods is

$$P_{COCR} = \frac{(2n_z + 11)t_{fl}N \ln 2}{4(t_s + t_w)} \quad (11)$$

and

$$P_{PCOCR} = \frac{(2n_z + 12)t_{fl}N \ln 2}{2t_s + 6t_w} \quad (12)$$

respectively. Since $t_s \gg t_w$, $P_{PCOCR} \approx 2P_{COCR}$ for any $n_z > 0$ Hence, the scalability of PCOCR method is better than that of COCR method.

It can also be obtained that the global communication improving rate of PCOCR against COCR is

$$\eta = \frac{T_{COCR} - T_{PCOCR}}{T_{COCR}} \approx \frac{2t_s P \log P + t_{fl}N}{4t_s P \log P + (2n_z + 11)t_{fl}N} \to 50\% \quad (13),$$

when N is fixed and P is large enough.

Iso-efficiency Analysis about Two Methods

This section presents a concept modeling the scalability of a parallel algorithm on a parallel computer. The concept is used to analyze a single iteration step of a parallel COCR-like iterative method.

It is known that sequential algorithms traditionally are evaluated in terms of their execution time. The sequential execution time is usually expressed as a function of a free variable called problem size. In this paper, it is concerned about COCR-like iterative methods. Since it is not known in advance how many iteration steps a method needs to converge, the whole algorithm is excluded from consideration until its termination, but a single iteration step is considered, and N, the dimension of the coefficient matrix, is taken as the problem size. The execution time of the fastest known sequential algorithm to perform a single COCR-like iteration is

$$T_{seq}(N) = cNt_{fl} = \theta(\mathrm{N}) \quad (14)$$

Where c is a constant and t_{fl} is the time required to perform a floating point operation. For the motivation of the iso-efficiency concept, we briefly state the conventional definitions of speedup and efficiency. The speedup S is defined as the ratio of the time to solve a problem on a single processor using the fastest known sequential algorithm to the time required to solve the same problem on a parallel computer, $S = T_{seq} / T_{par}$. The efficiency E is defined as the ration of the speedup to the number of processors, that $E = S / p$. The optimal speedup is equal to P and the corresponding efficiency is equal to one. One can expect to keep efficiency constant by allowing T_{seq} to grow properly with increasing number of processors. The rate at which T_{seq} has to be increased with respect to the number of processors to maintain a fixed efficiency can serve as a measure of scalability.

Algorithm implementations on real parallel computers do not achieve optimal speedup. For example, data communication delays and synchronization are reasons for non-optimal speedup. All causes of dropping the theoretically ideal speedup are called overhead and the total overhead function is formally defined as

$$T_{over}(N,P) = PT_{par}(N,P) - T_{seq}(N) \quad (15)$$

i.e., that part of the total time spent in solving a problem summed over all processors PT_{par} that is not incurred by the fastest known sequential algorithm T_{seq}. So, the efficiency can be expressed as a function of the total overhead and the execution time of the fastest known sequential algorithm

$$\begin{aligned} E &= \frac{S}{P} = \frac{T_{seq}(N)}{PT_{par}(N,P)} = \frac{T_{seq}(N)}{T_{seq} + T_{over}(N,P)} \\ &= \frac{1}{1 + T_{over}(N,P) / T_{seq}(N)} \end{aligned} \quad (16)$$

The rate with respect to P at which T_{seq} has to be

increased to keep efficiency constant is used to show the quality of a scalable parallel system. For example, if T_{seq} has to be increased as an exponential function of P to maintain efficiency fixed, the system is poorly scalable. A system is highly scalable if one only has to linearly increase T_{seq} with respect to P. Such growth rates can be calculated from (16) or from

$$T_{seq}(N) = \frac{E}{1-E} T_{over}(N,P) \quad (17),$$

where E is the desired efficiency to be maintained. Rather than deriving a growth rate of T_{seq} with respect to P yielding an iso-efficiency function, we are concerned about analyzing how the problem size N has to be increased with respect to P for keeping the efficiency from dropping. The task is therefore to solve (17) for N as a closed function about P.

The iso-efficiency analysis is carried out for a single COCR-like iteration step. To calculate growth rates from (17), we need to know T_{seq} and T_{over} of a single COCR-like iteration step. The total execution time of the fastest given by (14). The total overhead is solely due to communication times, i.e., $T_{over} = PT_{comm}$.

Then COCR is compared with PCOCR methods by analysis. From Table 1, we can get the sequential time for COCR method per iteration

$$\begin{aligned} T_{COCR}^{seq} &= 4t_{vec\_upd}^{comp} + 2t_{inn\_prod}^{comp}(1) + t_{mat\_vec}^{comp} \\ &= (2n_z + 11)t_{fl} N \end{aligned} \quad (18)$$

The communication time for COCR method per iteration

$$\begin{aligned} T_{COCR}^{comm} &= 2t_{inn\_prod}^{comp}(1) + t_{mat\_vec}^{comm} \\ &= 4(t_s + t_w)\log P + 2n_m t_s + 2(2n_b + n_m)t_w \end{aligned} \quad (19)$$

The total overhead for COCR method per iteration

$$T_{COCR}^{over} = PT_{COCR}^{par} - T_{COCR}^{seq} = PT_{COCR}^{comm} \quad (20)$$

Inserting (18) and (20) into (17), we can get following equation:

$$\begin{aligned} T_{COCR}^{seq} &= \frac{E}{1-E} T_{COCR}^{over} \\ N_{COCR} &= \frac{4(t_s + t_w)E}{t_{fl}(2n_z + 11)(1-E)} P\log P \\ &\approx \frac{4t_s E}{t_{fl}(2n_z + 11)(1-E)} P\log P \end{aligned} \quad (21)$$

We can also get the sequential time for PCOCR method per iteration

$$\begin{aligned} T_{PCOCR}^{comm} &= 4t_{vec\_upd}^{comp} + 3t_{inn\_prod}^{comp}(1) + t_{mat\_vec}^{comp} \\ &= (2n_z + 12)t_{fl} N \end{aligned} \quad (22)$$

The communication time for PCOCR method per iteration

$$\begin{aligned} T_{PCOCR}^{comm} &= t_{inn\_prod}^{comm}(3) + t_{mat\_vec}^{comm} = (2t_s + 6t_w)\log P \\ &\quad + 2n_m t_s + 2(2n_b + n_m)t_w \end{aligned} \quad (23)$$

The total overhead for PCOCR method per iteration

$$T_{PCOCR}^{over} = PT_{PCOCR}^{par} - T_{PCOCR}^{seq} = PT_{PCOCR}^{comm} \quad (24)$$

Inserting (22) and (24) into (17), we can get following equation:

$$\begin{aligned} T_{PCOCR}^{seq} &= \frac{E}{1-E} T_{PCOCR}^{over} \\ N_{PCOCR} &= \frac{(2t_s + 6t_w)E}{t_{fl}(2n_z + 12)(1-E)} P\log P \\ &\approx \frac{2t_s E}{t_{fl}(2n_z + 12)(1-E)} P\log P \end{aligned} \quad (25),$$

where T_{PCOCR}^{seq} and T_{COCR}^{seq} are sequential runtime sequential runtime for PCOCR and COCR respectively, while T_{PCOCR}^{comm} and T_{COCR}^{comm} are communication time respectively, T_{PCOCR}^{over} and T_{COCR}^{over} are overhead time respectively.

From equations (21) and (25), it can be seen that PCOCR method has better parallelism and scalability than COCR method.

Numerical Experiments

In this section, we report some numerical experiments with PCOCR and COCR; then evaluate both two methods in aspects of the number of iterations (Its) and log10 of true relative residual 2-norm (TRR) defined as $\log_{10} \| b - Ax_n \| / \| b \|$. All tests are performed on a x64 PC equipped with Intel Core i7-2600(4 cores, 3.40 GHz, 4 GB). Codes are written in Matlab (R2009a). In all cases, the iteration is started with $x_0 = 0$, and the stopping criterion is $\| r_n \| / \| b \| \le 10^{-6}$. The convergence plots show $\log_{10}$ of the relative residual 2-norm, $\log_{10} \| r_n \| / \| b \|$, (on the vertical axis) versus Its (on the horizontal axis).

TABLE 2 THE NUMERICAL RESULTS FOR TWO MATRICES

Matrix	N	Its		TRR	
		PCOCR	COCR	PCOCR	COCR
COCR	324	1397	1444	9.89e-7	8.40e-7
PCOCR	841	410	408	9.78e-7	9.67e-7

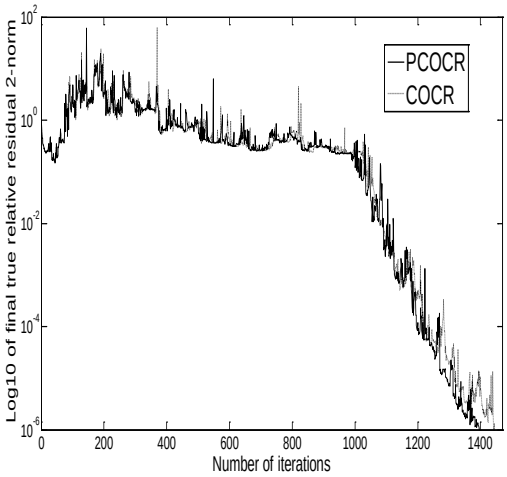

FIG. 1 THE CONVERGENCE HISTORY OF QC324

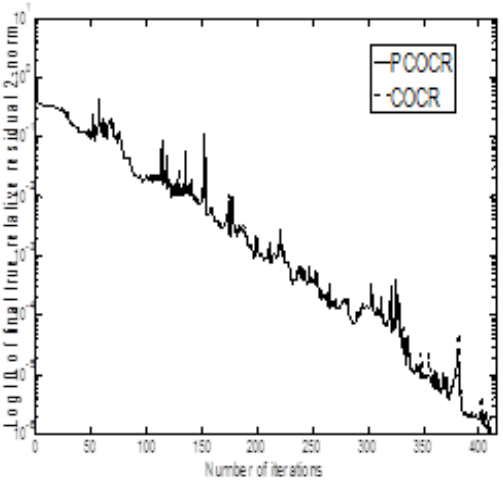

FIG. 2 THE CONVERGENCE HISTORY OF YOUNG1C

We chose two complex symmetric matrices downloaded from Matrix Market from quantum chemistry (QC324) and acoustic scattering (YOUNG1C) respectively. Numerical results for each test problem are given in Table 2, where N is order of matrix. The right-hand side b is chosen as $(1+i,...,1+i)^T$. Convergence histories for QC324 and YOUNG1c are shown in FIG. 1 and 2. It is indicated that PCOCR and COCR give almost the same convergence behavior. At the same time, the two methods also give almost the same accuracy on TRR at each final iteration step (see TABLE 2). All these numerical results show that PCOCR and COCR have almost the same numerical stability.

Conclusions

In this paper, a parallel COCR method for solving complex symmetric linear systems is proposed for distributed parallel architectures, which reduces two global synchronization points to one by eliminating corresponding data dependency in the original COCR method; as well keeps the number of vector updates and matrix-vector multiplications constant. Performance analysis and numerical results show that PCOCR method not only has better parallelism and scalability than the original COCR method, but also has the same numerical stability.

ACKNOWLEDGMENT

The authors would like to thank the referees and Editor for their helpful and detailed suggestions for revising this manuscript.

The project is partly supported by the NSFC of China (No. 61202098, 61170309, 91130024 and 11171039) and Aeronautical Science Foundation of China (2013ZD55006).

REFERENCES

B¨cker H. M. and Sauren M., A parallel version of the quasi-minimal residual method based on coupled two-term recurrences. In Proceedings of Workshop on Applied Parallel Computing in Industrial Problems and Optimization (Para96). Technical University of Denmark, Lyngby, Denmark, Springer-Verlag, August 1996.

Collignon T. P. and vanGijzen M. B., Fast solution of nonsymmetric linear systems on Grid computers using parallel variants of IDR(s), Delft University of Technology,Reports of the Department of Applied Mathematical Analysis, Report 10-05, 2010.

Dongarra J. J., Duff I. S., Sorensen D. C., and van der Vorst H. A., Numerical Linear Algebra for High-Performance Computers, SIAM, Philadelphia, PA, 1998.

Freund R. W. and Nachtigal N. M., QMR: a quasi-minimal residual method for non-Hermitian linear systems. Numerische Mathematik, 60, 315-339, 1991.

Gu T. X., Liu X. P. and Mo Z. Y., Multiple search direction conjugate gradient method I: methods and their propositions, Int. J. Comput. Math., 81(9), 1133-1143, 2004.

Liu J., Liu X. P., Chi L. H., Hu Q. F. and Li X. M., An Improved Conjugate Residual algorithm for large symmetric linear systems, In Proceedings of ICCP6, Edited by Xi-Jun Yu, 2004.

Gu T. X., Zuo X. Y., Zhang L. T., Zhang W. Q. , and Sheng Z. Q., An improved bi-conjugate residual algorithm suitable

for distributed parallel computing, Applied Mathematics and Computation , 186, 1243-1253, 2007.

Saad Y., Iterative Methods for Sparse Linear Systems, PWS Publishing Company, Boston, 1996.

Sogabe T. and Zhang S. L., Extended Conjugate Residual Methods for Solving Non- symmetric Linear Systems, In Numerical Linear Algebra and Optimization, Edited by Ya-Xiang Yuan, Science Press, Beijing/New York, pp88-99, 2003.

Sogabe T. and Zhang S. L., A COCR method for solving complex symmetric linear systems, In Journal of Computational and Applied Mathematics, 199, 297-303, 2007.

de Sturler E. and van der Vorst H. A., Reducing the effect of the global communication in GMRES(m) and CG on parallel distributed memory computers. Applied Numerical Mathematics, 18, 441-459, 1995.

de Sturler E., A performance model for Krylov subspace methods on mesh-based parallel computers, Parallel Computing, 22, 57-74, 1996.

van der Vorst H. A., Bi-CGSTAB: a fast and smoothly converging variant of bi-CG for the solution of nonsymmetric linear systems, SIAM J. Sci. Stat. Comput., 13,631-644, 1992.

van der Vorst H. A., Melissen J. B., A Petrov-Galerkin type method for solving Ax = b, where A is symetric complex, IEEE Trans. Mag., 26(2), 706-708, 1990.

Boisvert R., Pozo R., Remington K., B. Miller and R. Lipman, The Matrix Market, 2000, http://math.nist.gov/MatrixMarket/

Yang T. R. and Lin H. X., The improved quasi-minimal residual method on massively distributed memory computers. In Proceedings of the International Conference on High Performance Computing and Networking (HPCN-97), April 1997.

Yang T. R., The improved CGS method for large and sparse linear systems on bulk synchronous parallel architectures, 2002 5th Intern. Conf. Algorithms and Architec-tures for Parallel Processing, IEEE Computer Society, pp. 232-237, 2002.

Yang T. R. and Brent R. P., The improved BICGSTAB method for large and sparse unsymmetric linear systems on parallel distributed memory architectures, 2002 5th Intern. Conf. Algorithms and Architectures for Parallel Processing, IEEE Computer Society, pp. 324-328, 2002.

Yang T. R. and Brent R. P., The Improved BiCG Method for Large and Sparse Linear Systems on Parallel Distributed Memory Architectures, Information Journal,6, 349-360, 2003.

Zuo X. Y., Gu T. X. and Mo Z. Y., An improved GPBi-CG algorithm suitable for distributed parallel computing, Applied Mathematics and Computation , 215, 4101-4109, 2010.