Comp. Appl. Math.
DOI 10.1007/s40314-014-0206-z

A parallel version of GPBi-CG method suitable for distributed parallel computing

Xian-Yu Zuo · Li-Tao Zhang · Tong-Xiang Gu · Feng-Bin Zheng · Ning Li

Received: 7 August 2014 / Revised: 3 December 2014 / Accepted: 8 December 2014

Abstract In this paper, one new parallel version of GPBi-CG method (PGPBi-CG method, in brief) is proposed for solving large sparse linear systems with unsymmetrical coefficient matrices on distributed parallel environments. The method reduces three global synchronization points to one by reconstructing GPBi-CG method and the communication time required for the inner product can be efficiently overlapped with computation time of vector updates. It combines the elements of numerical stability with the characteristics of design of parallel algorithms. The cost is only slightly increased computation time, which can be ignored, compared with the reduction of communication time. Performance and isoefficiency analysis shows that the PGPBi-CG method has better parallelism and scalability than the GPBi-CG method. Numerical experiments show that the scalability can be improved by a factor 3 and the improvement in parallel communication performance approaches 66.7 %.

Communicated by Philippe Devloo.

This research of this author is supported by the NSFC (61202098, 61203179, 61170309, 11171039, 11471098 and 61272544), China Postdoctoral Science Foundation (2014M552001), Henan Province Postdoctoral Science Foundation (2013031), Aeronautical Science Foundation of China (2013ZD55006), Project of Youth Backbone Teachers of Colleges and Universities in Henan Province (2013GGJS-142), Major Project of development foundation of science and technology of CAEP (2012A0202008).

X.-Y. Zuo (✉) · F.-B. Zheng (✉) · N. Li
Institute of Data and Knowledge Engineering, School of Computer and Information Engineering, Henan University, Kaifeng, Henan 475004, People's Republic of China
e-mail: xianyu_zuo@163.com

F.-B. Zheng
e-mail: zhengfb@henu.edu.cn

L.-T. Zhang
Department of Mathematics and Physics, Zhengzhou Institute of Aeronautical Industry Management, Zhengzhou, Henan 450015, People's Republic of China

T.-X. Gu
Laboratory of Computationary Physics, Institute of Applied Physics and Computational Mathematics, P.O.Box 8009, Beijing 100088, People's Republic of China

Published online: 31 December 2014

Keywords PGPBi-CG method · Krylov subspace · Sparse unsymmetrical linear systems · Global communication · Distributed parallel environments

Mathematics Subject Classification 65F10 · 65F15

1 Introduction

Since scientific and engineering problems arising from “real” applications are typically becoming larger and larger, there is a strong demand for ever increasing computing power. Besides the tremendous progress in computer architecture, the main feature of delivering high performance is the design of sophisticated algorithms. When using parallel computing platforms, the need for algorithms specifically designed for parallel processing is particularly important because, in addition to computation, communication among the processors has to be controlled.

One of the fundamental tasks of numerical computation is to solve linear systems. These systems arise very frequently in scientific and engineering computing. For example, they comes from finite difference or finite element approximations to PDEs, from intermediate steps in computing the solution of nonlinear problems or from subproblems in linear and nonlinear programming. Usually, these linear systems are large, sparse and nonsymmetric and solved by iterative methods (Saad 1996).

Among the iterative methods for large sparse systems, Krylov subspace methods are the most powerful. The conjugate gradient (CG) method is used for solving symmetric positive definite linear systems, the GMRES method, BiCG method (Saad 1996), BiCGStab method (van der Vorst 1992), GPBi-CG method (Zhang 1997), QMR method (Freund and Nachtigal 1991), BiCR method (Sogabe and Zhang 2003), IDR(s) method (Sonneveld and van Gijzen 2008), and the cooperative conjugate gradient method (Bhaya et al. 2012) for solving nonsymmetric linear systems and so on are all examples.

The basic time-consuming computational kernels of all Krylov subspace methods, involving GPBi-CG method, are usually (Saad 1996): 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 operation, we still cannot 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 only between nearby processors. The bottleneck is usually due to inner products enforcing global communication. 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. A detailed discussion on the communication problem on distributed memory systems can be found in de Sturler and van der Vorst (1995), de Sturler (1996).

Three methods can be used to solve the bottleneck which leads to performance degeneration. The first is to eliminate data dependency, so that several inner products can be computed and passed at the same time. The second is reconstructing algorithm, resulting communication and computation can be overlapped efficiently. The last is replacing of computation involving global communications by the other computation without global communications. Of course, they can be used concurrently.

Bücker and Sauren (1996) and Yang and Lin (1997) proposed new parallel Quasi-Minimal Residual(QMR) method based on coupled two-term recurrences Lanczos process. Sturler and Vorst (1995) gave how to reduce effect of the global communication in GMRES(m) and CG methods. Yang (2002) and Yang and Brent (2002, 2003) proposed the improved CGS, BiCG and BiCGStab methods, respectively. Liu et al. (2005) gave an improved CR algorithm. Gu et al. (2005, 2007, 2008) propose an improved BiCR method, an improved BiCGSTAB(2) method and the parallel QMRCGSTAB method, Collignon and Van Gijzen (2010) propose the parallel variants of IDR(s)method. All these methods depend on the former two strategies. Gu et al. (2004a, b) propose 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. Grigori and Moufawad (2013) proposed a communication avoiding ILU0 preconditioner using communication avoiding method.

Based on the former two remedies, we would like to propose a parallel version of GPBi-CG method. 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 slightly increased computation. Performance and isoefficiency analysis show that PGPBi-CG method has better parallelism and scalability than GPBi-CG method. Especially, the parallel performance can be improved by a factor 3.

The paper is organized as follows. Section 2 presents the algorithm design of PGPBi-CG method. Performance analysis about the two methods is given in Sect. 3. In Sect. 4, isoefficiency analysis about the two methods is presented. Numerical experiments for comparing two methods which carried out on distributed memory parallel machine are reported in Sect. 5. Finally, we make some concluding remarks in Sect. 6.

2 Algorithm design of PGPBi-CG method

Consider solving a large sparse unsymmetrical linear system

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

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

Assuming matrices and corresponding vectors have been distributed to each processor according to row or domain decomposition, and have perfect load balance; Matrix operations are well sparse-structured and matrix–vector multiplications can be implemented with communication only between nearby processors. The important bottleneck for Krylov subspace methods is usually due to inner products enforcing global communication.

For comparison, we give the GPBi-CG method (Zhang 1997), 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 [GPBi-CG (Zhang 1997)]

```
(1)   $x_0$ is an initial guess, $r_0 = b - Ax_0$, $r_0^*$ is an arbitrary vector, such
      that $(r_0^*, r_0) \neq 0$, e.g., $r_0^* = r_0$, and set $t_{-1} = w_{-1} = 0$, $\beta_{-1} = 0$,
(2)   For $n = 0, 1, \ldots$ until $\|r_n\| \leq \varepsilon \|b\|$ do :
(3)       $p_n = r_n + \beta_{n-1}(p_{n-1} - u_{n-1})$,
(4)    $\alpha_n = \frac{(r_0^*, r_n)}{(r_0^*, Ap_n)}$,
(5)    $t_n = r_n - \alpha_n Ap_n$,
(6)    $y_n = t_{n-1} - t_n - \alpha_n w_{n-1}$,
(7)    $\zeta_n = \frac{(y_n,y_n)(At_n,t_n)-(y_n,t_n)(At_n,y_n)}{(At_n,At_n)(y_n,y_n)-(y_n,At_n)(At_n,y_n)}$,
       $\eta_n = \frac{(At_n,At_n)(y_n,t_n)-(y_n,At_n)(At_n,t_n)}{(At_n,At_n)(y_n,y_n)-(y_n,At_n)(At_n,y_n)}$,
       (if $n = 0$, then $\zeta_n = \frac{(At_n,t_n)}{(At_n,At_n)}$, $\eta_n = 0$)
(8)    $u_n = \zeta_n Ap_n + \eta_n(t_{n-1} - r_n + \beta_{n-1}u_{n-1})$,
(9)    $z_n = \zeta_n r_n + \eta_n z_{n-1} - \alpha_n u_n$,
(10)   $x_{n+1} = x_n + \alpha_n p_n + z_n$,
(11)   $r_{n+1} = t_n - \eta_n y_n - \zeta_n At_n$,
(12)   $\beta_n = \frac{\alpha_n}{\zeta_n} \frac{(r_0^*, r_{n+1})}{(r_0^*, r_n)}$,
(13)   $w_n = At_n + \beta_n Ap_n$.
```

In Algorithm 1, steps (4), (7) and step (12) require inner products, and other computing steps are closely connected with these inner products. Furthermore, steps (4), (7) and step (12) have close data dependency. So there are three 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. In order to reduce this negative effect, we can use one global communication to accomplish all inner products in one iteration step.

The main idea of PGPBi-CG algorithm is to eliminate data dependency of inner products computing between steps (4), (7) and step (12) through mathematical derivation.

Define that $f_0 = A^T r_0^*, \delta_n = (f_0, p_n), b_n = (f_0, r_n), c_n = (f_0, u_{n-1}), a_1 = (r_0^*, t_n), a_2 = (r_0^*, y_n), d_n = (r_0^*, At_n),\ e_1 = (f_0, Ap_n), e_2 = (f_0, y_n), e_3 = (f_0, At_n)$, step (4) and (12) in GPBi-CG method become

$$\delta_n = (f_0, p_n) = (f_0, r_n + \beta_{n-1}(p_{n-1} - u_{n-1})) = b_n + \beta_{n-1}(\delta_{n-1} - c_n);$$
$$\rho_{n+1} = (r_0^*, r_{n+1}) = a_1 - \eta_n a_2 - \zeta_n d_n;$$

while,we can get

$$c_{n+1} = \zeta_n e_1 + \eta_n(d_{n-1} - b_n + \beta_{n-1}c_n);$$
$$b_{n+1} = d_n - \eta_n e_2 - \zeta_n e_3;$$

After these derivations, we can reconstruct a parallel version of the GPBi-CG method, which we call it PGPBi-CG method. The new method can be presented in algorithmic form as follows:

Algorithm 2 (PGPBi-CG)

```
(1)    $x_0$ is an initial guess, $r_0 = b - Ax_0$, $r_0^*$ is an arbitrary vector, such
       that $(r_0^*, r_0) \neq 0$, e.g., $r_0^* = r_0$, and set $t_{-1} = w_{-1} = 0$, $\beta_{-1} = 0$,$\rho_0 =$
       $(r_0^*, r_0)$,$f_0 = A^T r_0^*$, $b_0 = (f_0, r_0^*)$,
(2)    For $n = 0, 1, \ldots$ until $\|r_n\| \leq \varepsilon \|b\|$ do :
(3)        $p_n = r_n + \beta_{n-1}(p_{n-1} - u_{n-1})$,
(4)        $\delta_n = b_n + \beta_{n-1}(\delta_{n-1} - c_n)$,
(5)        $\alpha_n = \frac{\rho_n}{\delta_n}$,
(6)        $t_n = r_n - \alpha_n Ap_n$,
(7)        $y_n = t_{n-1} - t_n - \alpha_n w_{n-1}$,
(8)        $a_1 = (r_0^*, t_n)$,$a_2 = (r_0^*, y_n)$,$e_1 = (f_0, Ap_n)$,
           $e_2 = (f_0, y_n)$, $e_3 = (f_0, At_n)$, $d_n = (r_0^*, At_n)$,
```

Table 1 The amount of calculation per iteration

Methods	Vector update	Mat–vec	Inner product	Global synchro.
GPBi-CG	15	2	7	3
PGPBi-CG	15	2	11	1

$\zeta_n = \frac{(y_n,y_n)(At_n,t_n)-(y_n,t_n)(At_n,y_n)}{(At_n,At_n)(y_n,y_n)-(y_n,At_n)(At_n,y_n)}$,
$\eta_n = \frac{(At_n,At_n)(y_n,t_n)-(y_n,At_n)(At_n,t_n)}{(At_n,At_n)(y_n,y_n)-(y_n,At_n)(At_n,y_n)}$,
(if $n = 0$, then $\zeta_n = \frac{(At_n,t_n)}{(At_n,At_n)}$, $\eta_n = 0$)
(9) $u_n = \zeta_n Ap_n + \eta_n(t_{n-1} - r_n + \beta_{n-1}u_{n-1})$,
(10) $c_{n+1} = \zeta_n e_1 + \eta_n(d_{n-1} - b_n + \beta_{n-1}c_n)$,
(11) $z_n = \zeta_n r_n + \eta_n z_{n-1} - \alpha_n u_n$,
(12) $x_{n+1} = x_n + \alpha_n p_n + z_n$,
(13) $r_{n+1} = t_n - \eta_n y_n - \zeta_n At_n$,
(14) $b_{n+1} = d_n - \eta_n e_2 - \zeta_n e_3$,
(15) $\rho_{n+1} = a_1 - \eta_n a_2 - \zeta_n d_n$,
(16) $\beta_n = \frac{\rho_{n+1}}{\rho_n}$,
(17) $w_n = At_n + \beta_n Ap_n$.

The inner products in step (8) are independent of each other. Therefore, they need only one global synchronizations.

3 Performance analysis of both methods

The derivation of the PGPBi-CG method shows that PGPBi-CG and GPBi-CG methods are mathematically equivalent. The amount of calculation per iteration for each method without preconditioning is indicated in Table 1.

From Table 1, we can see that PGPBi-CG method needs four inner productions more than GPBi-CG method. But the global synchronization points per iteration have been reduced from three to two. The increased amount of computation is small relative to the reduction of global communication, and this can also been seen in the numerical experiments of next section.

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

Similar to de Sturler (1996), we give a performance analysis of both methods on distributed memory parallel computers. 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, we only consider time complexity of parallel computation and communication of one iteration.

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

The global accumulation and broadcast time for one inner product are taken as $2\log P(t_{\mathrm{s}}+t_{\mathrm{w}})$, while the global accumulation and broadcast for k simultaneous inner products take $2\log P(t_{\mathrm{s}}+kt_{\mathrm{w}})$. Assume coefficient matrix is mapped to processors such that the matrix–vector needs a processor which only communicates with the nearest neighbor processors. 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 $2(2n_{\mathrm{b}}+n_{\mathrm{m}})$ per processor. For both methods, the communication time of one matrix–vector product is $2n_{\mathrm{m}}t_{\mathrm{s}}+2(2n_{\mathrm{b}}+n_{\mathrm{m}})t_{\mathrm{w}}$.

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

$$t_{\mathrm{vec\_upd}} = 2t_{\mathrm{fl}}N/P \tag{2}$$

the time for k simultaneous inner products is

$$t_{\mathrm{inn\_prod}}(k) = 2kt_{\mathrm{fl}}N/P + 2\log P(t_{\mathrm{s}}+kt_{\mathrm{w}}) \tag{3}$$

and the time for a matrix–vector is

$$t_{\mathrm{mat\_vec}} = (2n_{\mathrm{z}}-1)t_{\mathrm{fl}}N/P + 2n_{\mathrm{m}}t_{\mathrm{s}} + 2(2n_{\mathrm{b}}+n_{\mathrm{m}})t_{\mathrm{w}} \tag{4}$$

From Table 1, the time per iteration of GPBi-CG method is

$$\begin{aligned} T_{\mathrm{GPBi\text{-}CG}} &= 15t_{\mathrm{vec\_upd}} + 2t_{\mathrm{inn\_prod}}(1) + t_{\mathrm{inn\_prod}}(5) + 2t_{\mathrm{mat\_vec}} \\ &= (4n_{\mathrm{z}}+42)t_{\mathrm{fl}}N/P + 4\log P(t_{\mathrm{s}}+t_{\mathrm{w}}) + 2\log P(t_{\mathrm{s}}+5t_{\mathrm{w}}) \\ &\quad + 4n_{\mathrm{m}}t_{\mathrm{s}} + 4(2n_{\mathrm{b}}+n_{\mathrm{m}})t_{\mathrm{w}} \end{aligned} \tag{5}$$

and of PGPBi-CG is

$$\begin{aligned} T_{\mathrm{PGPBi\text{-}CG}} &= 15t_{\mathrm{vec\_upd}} + +t_{\mathrm{inn\_prod}}(11) + 2t_{\mathrm{mat\_vec}} \\ &= (4n_{\mathrm{z}}+46)t_{\mathrm{fl}}N/P + 2\log P(t_{\mathrm{s}}+11t_{\mathrm{w}}) \\ &\quad + 4n_{\mathrm{m}}t_{\mathrm{s}} + 4(2n_{\mathrm{b}}+n_{\mathrm{m}})t_{\mathrm{w}} \end{aligned} \tag{6}$$

We know that $t_{\mathrm{s}} >> t_{\mathrm{w}}$ for massively distributed parallel computer. Compare Eq. (5) and (6), we get that the parallelism of PGPBi-CG method is better than that of GPBi-CG method since $T_{\mathrm{PGPBi\text{-}CG}} < T_{\mathrm{GPBi\text{-}CG}}$.

Minimizing $T_{\mathrm{GPBi\text{-}CG}}$ and $T_{\mathrm{PGPBi\text{-}CG}}$ from (5) and (6), we obtain that the number of processors for minimal parallel time of both methods is

$$P_{\mathrm{GPBi\text{-}CG}} = \frac{(4n_{\mathrm{z}}+42)t_{\mathrm{fl}}N\ln 2}{4(t_{\mathrm{s}}+t_{\mathrm{w}})+2(t_{\mathrm{s}}+5t_{\mathrm{w}})} = \frac{(2n_{\mathrm{z}}+21)t_{\mathrm{fl}}N\ln 2}{3t_{\mathrm{s}}+7t_{\mathrm{w}}} \tag{7}$$

and

$$P_{\mathrm{PGPBi\text{-}CG}} = \frac{(4n_{\mathrm{z}}+46)t_{\mathrm{fl}}N\ln 2}{2(t_{\mathrm{s}}+11t_{\mathrm{w}})} = \frac{(2n_{\mathrm{z}}+23)t_{\mathrm{fl}}N\ln 2}{t_{\mathrm{s}}+11t_{\mathrm{w}}}, \tag{8}$$

respectively. Since $t_{\mathrm{s}} >> t_{\mathrm{w}}$, $\frac{P_{\mathrm{PGPBi\text{-}CG}}}{P_{\mathrm{GPBi\text{-}CG}}} \approx 3 + \frac{6}{2n_{\mathrm{z}}+21} > 3$ for any $n_{\mathrm{z}} > 0$. Hence, the scalability of PGPBi-CG method is better than that of GPBi-CG method.

We can also obtain that improving rate of PGPBi-CG against GPBi-CG is

$$\eta = \frac{T_{\text{GPBi-CG}} - T_{\text{PGPBi-CG}}}{T_{\text{GPBi-CG}}} \approx \frac{4t_s P \log P - 4t_{\text{fl}} N}{6t_s P \log P + (4n_z + 42)t_{\text{fl}} N} \rightarrow 66.7\ \% \tag{9}$$

for $t_s >> t_w$, when N is fixed and P is large enough.

4 Isoefficiency 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 GPBi_CG-like iterative method.

We know that sequential algorithms are traditionally evaluated in term of their execution time. The sequential execution time is usually expressed as a function of a free variable called *problem size*. In this paper, we are concerned with GPBi_CG-like iterative methods. Since it is not known in advance how many iteration steps a method needs to converge, we do not consider the whole algorithm until its termination but consider a single iteration step and take N,the dimension of the coefficient matrix, as the problem size. The execution time of the fastest known sequential algorithm to perform a single GPBi_CG-like iteration is

$$T_{\text{seq}}(N) = cNt_{\text{fl}} = \Theta(N) \tag{10}$$

where c is a constant and t_{fl} is the time required to perform a floating point operation.

For the motivation of the isoefficiency 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, $S = T_{\text{seq}}/T_{\text{par}}$. The efficiency E is defined as the ration of the speedup to the number of processors, $E = S/P$. 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 nonoptimal speedup. All causes of dropping the theoretically ideal speedup are call overhead and the *total overhead function* is formally defined as:

$$T_{\text{over}}(N, P) = PT_{\text{par}}(N, P) - T_{\text{seq}}(N) \tag{11}$$

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

$$E = \frac{S}{P} = \frac{T_{\text{seq}}(N)}{PT_{\text{par}}(N, P)} = \frac{T_{\text{seq}}(N)}{T_{\text{seq}} + T_{\text{over}}(N, P)} = \frac{1}{1 + T_{\text{over}}(N, P)/T_{\text{seq}}(N)} \tag{12}$$

The rate with respect to P at which T_{seq} has to be increased to keep efficiency constant is used to asses 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 only one has to linearly increase T_{seq} with respect to P. Such growth rates can be calculated from (12) or from

$$T_{\mathrm{seq}}(N) = \frac{E}{1-E} T_{\mathrm{over}}(N, P), \tag{13}$$

where E is the desired efficiency to be maintained. Rather than deriving a growth rate of T_{seq} with respect to P yielding an*isoefficiency function*, we are concerned with analyzing how the problem size N has to be increased with respect to P to keep the efficiency from dropping. The task is, therefore, to solve (13) for N as a closed function of P.

We carried out the isoefficiency analysis for a single GPBi_CG-like iteration step. To calculate growth rates from (13), we need to know T_{seq} and T_{over} of a single GPBi_CG-like iteration step. The total execution time of the fastest given by (10). The total overhead is solely due to communication times, i.e., $T_{\mathrm{over}} = PT_{\mathrm{comm}}$.

We compare GPBi-CG method with PGPBi-CG method by using isoefficiency analysis in the following context.

From Table 1, we can get the sequential time for GPBi-CG method per iteration

$$T^{\mathrm{seq}}_{\mathrm{GPBi\text{-}CG}} = 15t^{\mathrm{comp}}_{\mathrm{vec\_upd}} + 7t^{\mathrm{comp}}_{\mathrm{inn\_prod}}(1) + 2t^{\mathrm{comp}}_{\mathrm{mat\_vec}} = (4n_{\mathrm{z}} + 42)t_{\mathrm{fl}}N \tag{14}$$

The communication time for GPBi-CG method per iteration is

$$\begin{aligned} T^{\mathrm{comm}}_{\mathrm{GPBi\text{-}CG}} &= 2t^{\mathrm{comp}}_{\mathrm{inn\_prod}}(1) + t^{\mathrm{comp}}_{\mathrm{inn\_prod}}(5) + 2t^{\mathrm{comm}}_{\mathrm{mat\_vec}} \\ &= (6t_{\mathrm{s}} + 14t_{\mathrm{w}})\log P + 4n_{\mathrm{m}}t_{\mathrm{s}} + 4(2n_{\mathrm{b}} + n_{\mathrm{m}})t_{\mathrm{w}} \end{aligned} \tag{15}$$

The total overhead for GPBi-CG method per iteration is

$$T^{\mathrm{over}}_{\mathrm{GPBi\text{-}CG}} = PT^{\mathrm{par}}_{\mathrm{GPBi\text{-}CG}} - T^{\mathrm{seq}}_{\mathrm{GPBi\text{-}CG}} = PT^{\mathrm{comm}}_{\mathrm{GPBi\text{-}CG}} \tag{16}$$

Inserting (14) and (16) into (13), we can get following equation:

$$\begin{aligned} T^{\mathrm{seq}}_{\mathrm{GPBi\text{-}CG}} &= \frac{E}{1-E} T^{\mathrm{over}}_{\mathrm{GPBi\text{-}CG}} \\ N_{\mathrm{GPBi\text{-}CG}} &= \frac{(6t_{\mathrm{s}} + 14t_{\mathrm{w}})E}{t_{\mathrm{fl}}(4n_{\mathrm{z}} + 42)(1-E)} P\log P \approx \frac{3t_{\mathrm{s}}E}{t_{\mathrm{fl}}(2n_{\mathrm{z}} + 21)(1-E)} P\log P \end{aligned} \tag{17}$$

We can also get the sequential time for PGPBi-CG method per iteration

$$T^{\mathrm{seq}}_{\mathrm{PGPBi\text{-}CG}} = 15t^{\mathrm{comp}}_{\mathrm{vec\_upd}} + 9t^{\mathrm{comp}}_{\mathrm{inn\_prod}}(1) + 2t^{\mathrm{comp}}_{\mathrm{mat\_vec}} = (4n_{\mathrm{z}} + 46)t_{\mathrm{fl}}N \tag{18}$$

The communication time for PGPBi-CG method per iteration is

$$\begin{aligned} T^{\mathrm{comm}}_{\mathrm{PGPBi\text{-}CG}} &= t^{\mathrm{comm}}_{\mathrm{inn\_prod}}(11) + 2t^{\mathrm{comm}}_{\mathrm{mat\_vec}} \\ &= (2t_{\mathrm{s}} + 22t_{\mathrm{w}})\log P + 4n_{\mathrm{m}}t_{\mathrm{s}} + 4(2n_{\mathrm{b}} + n_{\mathrm{m}})t_{\mathrm{w}} \end{aligned} \tag{19}$$

The total overhead for PGPBi-CG method per iteration is

$$T^{\mathrm{over}}_{\mathrm{PGPBi\text{-}CG}} = PT^{\mathrm{par}}_{\mathrm{PGPBi\text{-}CG}} - T^{\mathrm{seq}}_{\mathrm{PGPBi\text{-}CG}} = PT^{\mathrm{comm}}_{\mathrm{PGPBi\text{-}CG}} \tag{20}$$

Inserting (18) and (20) into (13), we can get following equation:

$$\begin{aligned} T^{\mathrm{seq}}_{\mathrm{PGPBi\text{-}CG}} &= \frac{E}{1-E} T^{\mathrm{over}}_{\mathrm{PGPBi\text{-}CG}} \\ N_{\mathrm{PGPBi\text{-}CG}} &= \frac{(2t_{\mathrm{s}} + 22t_{\mathrm{w}})E}{t_{\mathrm{fl}}(4n_{\mathrm{z}} + 46)(1-E)} P\log P \approx \frac{t_{\mathrm{s}}E}{t_{\mathrm{fl}}(2n_{\mathrm{z}} + 23)(1-E)} P\log P \end{aligned} \tag{21}$$

where $T^{\mathrm{seq}}_{\mathrm{PGPBi\text{-}CG}}$ and $T^{\mathrm{seq}}_{\mathrm{GPBi\text{-}CG}}$ are sequential runtime sequential runtime for PGPBi-CG and GPBi-CG, respectively, while $T^{\mathrm{omm}}_{\mathrm{PGPBi\text{-}CG}}$ and $T^{\mathrm{comm}}_{\mathrm{GPBi\text{-}CG}}$ are communication time, respectively; $T^{\mathrm{over}}_{\mathrm{PGPBi\text{-}CG}}$ and $T^{\mathrm{over}}_{\mathrm{GPBi\text{-}CG}}$ are overhead time, respectively.

From Eqs. (17) and (21), We can see that PGPBi-CG method has has better parallelism and scalability than GPBi-CG method and the parallel performance can be improved by a factor 3. For different values of the efficiency, the results are shown in Figs. 1 and 2, where the filled color curves represent the theoretically derived isoefficiency function $N \sim O(P \log P)$, in which $t_s = 100\,\mu s$, $t_w = 20\,ns$, $t_{fl} = 10\,ns$ and $n_z = 9$ for our distributed memory parallel computer we do our numerical experiments on.

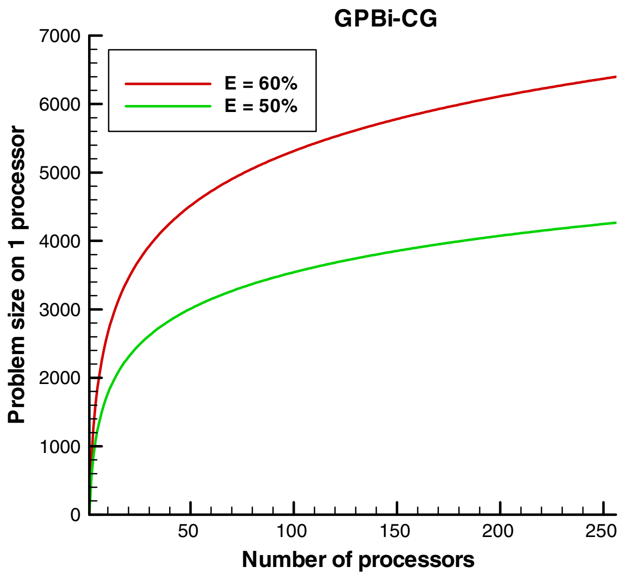

Fig. 1 Isoefficiency curve of GP-BiCG

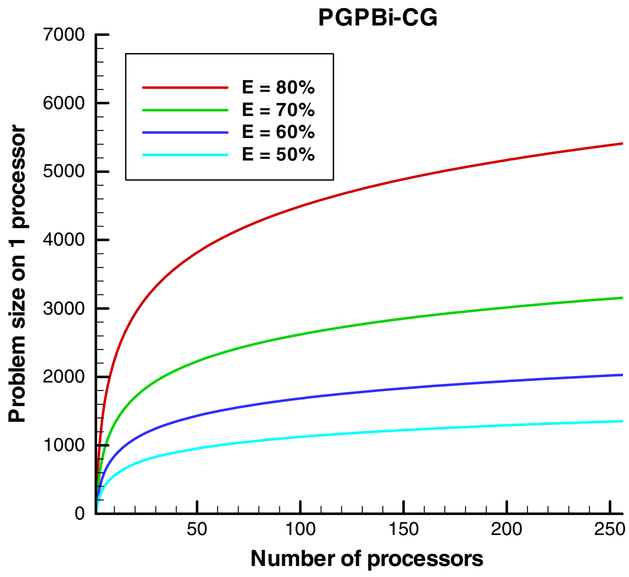

Fig. 2 Isoefficiency curve of PGPBi-CG

Table 2 Number of iterations for matrices from Matrix Market

Matrix	*N*	*NNZ*	Number of iterations	
			GPBi-CG	PGPBi-CG
ADD20	2,395	17,319	170	175
ADD32	4,960	23,884	34	34
BCSSTK02	66	2,211	20	20
BFW782A	782	7,514	257	195
CAVITY05	1,182	32,747	374	383
CDDE1	961	4,681	88	87
FS5414	541	4,285	1,003	1,004
ORSIRR2	886	5,970	1,076	980
PDE225	225	1,065	41	41
PDE900	900	4,380	78	79
PDE2961	2,961	14,585	152	147
SHERMAN1	1,000	3,750	307	301
SHERMAN5	3,312	20,793	2,195	2,204

5 Numerical experiments

5.1 Sequential experiments

Firstly, we present some numerical experiments which compare the sequential performance of GPBi-CG and PGPBi-CG methods without preconditioning in MATLAB 7.8(R2009a), and the corresponding matrices are from Matrix Market. The exact solution of a linear system of equations is $x = (1, 1, \dots, 1)^T$. In Table 2, we present the number of iterations necessary to fulfill the desired convergence criterion, where N and NNZ denote the dimensions of the corresponding matrices and the number of non-zero elements, respectively. The stopping criterion for successful convergence is less than 10^{-6} of the residual 2-norms. In all cases, the iteration was started with the initial solution vector $x_0 = 0$.

It can be seen from Table 2 that the PGPBi-CG and GPBi-CG methods have almost same convergence behavior. The difference is very small even if the number of iterations is different for some matrices.

5.2 Parallel experiments

In this section, we present some numerical results carried out on a MPP computer to compare PGPBi-CG with GPBi-CG. The methods solve the linear system arisen from nine-point difference discretization of the following partial differential equation:

$$a\frac{\partial^2 u}{\partial x^2} + b\frac{\partial^2 u}{\partial y^2} + c\frac{\partial u}{\partial x} + d\frac{\partial u}{\partial y} + eu + g = 0, \quad x, y \in (0, 1) \tag{22}$$

with the boundary conditions:

$$\begin{cases} \frac{\partial u}{\partial x}|_{x=0} = 0, & \frac{\partial u}{\partial x}|_{x=1} = 0 \\ u|_{y=0} = 0, & u|_{y=1} = 10 \end{cases}$$

Table 3 Comparison of performance

# CPU	# Unknowns	GPBi-CG			PGPBi-CG			Improvements	
		T_{G}	T_{G}^{c}	C_{G}	T_{PG}	T_{PG}^{c}	C_{PG}	η	η_c
4	14,400	2.97	0.31	10.44	3.25	0.12	11.71	−9.43	61.29
16	57,600	3.97	1.00	25.10	3.61	0.49	13.45	9.07	51.00
36	129,600	4.48	1.57	35.11	3.88	0.64	16.61	13.39	59.23
64	230,400	4.80	2.12	44.14	4.04	1.13	27.98	15.83	46.70
100	360,000	5.35	2.76	51.51	4.34	1.41	32.49	18.88	48.91
144	518,400	6.22	3.45	55.46	4.68	1.58	33.89	24.76	54.20
196	705,600	6.86	4.11	59.88	5.14	1.86	36.27	25.07	54.74
256	921,600	7.69	4.81	62.42	5.50	2.13	38.66	28.48	55.72
324	1,166,400	8.77	5.97	68.06	6.19	2.48	39.43	29.42	58.45
400	1,440,000	9.87	6.49	65.75	7.02	3.05	43.45	28.86	53.01
484	1,742,400	11.28	8.18	72.47	7.84	3.69	47.08	30.50	54.89

where $a = 50.0,\ b = 0.5,\ c = 0.001\sin(2\pi x),\ d = 0.001\sin(2\pi y),\ e = g = 0.0$.

Firstly, we compare the total wall clock time performance and communication performance of both methods for a fixed iteration of 2,000 and fixed unknowns of 3,600 on each processor. The results are listed in Table 3. The improvements' percentage (%) is obtain from $\eta = (1 - T_{\mathrm{PG}}/T_{\mathrm{G}}) \times 100$ and $\eta_{\mathrm{c}} = \left(1 - T_{\mathrm{PG}}^{\mathrm{c}}/T_{\mathrm{G}}^{\mathrm{c}}\right) \times 100$, where T_{PG} and T_{G} are wall clock time of PGPBi-CG and GPBi-CG method, while $T_{\mathrm{PG}}^{\mathrm{c}}$ and $T_{\mathrm{G}}^{\mathrm{c}}$ are their communication time, respectively. $C_{\mathrm{G}} = \left(T_{\mathrm{G}}^{\mathrm{c}}/T_{\mathrm{G}}\right) \times 100$ and $C_{\mathrm{PG}} = \left(T_{\mathrm{PG}}^{\mathrm{c}}/T_{\mathrm{PG}}\right) \times 100$ are percentages of communication time against wall clock time.

We can see that the communication time can be improved by a factor of about 3 from Table 3, which meets to our performance analysis. And communication time is dominating for GPBi-CG when 100 processors are used, while it is not dominating for PGPBi-CG up to 484 processors. We can conclude that PGPBi-CG method has better scalability than GPBi-CG method. As the number of processors increases, the bottle neck of global communication for GPBi-CG becomes serious. The improvement for communication becomes significant and tends to our theoretical rate 66.67 %. By the way, the improvement for wall clock time is negative on 4 processors. This is because the increase in computation is large against the decrease in communication for the low number CPUs used. Note that our PGPBi-CG method is designed for distributed parallel environment with large amount of processors or MPP systems.

Secondly, The parallel speed-ups of both methods are given in Fig. 3 (based on one processor) and Fig. 4 (based on 64 processors). These results are based on timing measurements on a 1,200 × 1,200 mesh grid and 1,000 iteration steps for each method. The speedup is computed as the ratio of the parallel execution time and the execution time using 1 and 64 processors. From the results, we can see clearly that the PGPBi-CG method can achieve much better parallel performance with a higher scalability than GPBi-CG method. By the way, we can see that the superlinear speed-up occurs. The reason is that the cache is utilized more efficiently for more processors, since the mesh is fixed. As the amount of computation becomes small, communication becomes dominant, and the superlinear speed-up disappears.

Thirdly, we compare the convergence of PGPBi-CG with GPBi-CG method on 16 and 64 processors (see Fig. 5) and on 324 and 400 processors (see Fig. 6). The toler-

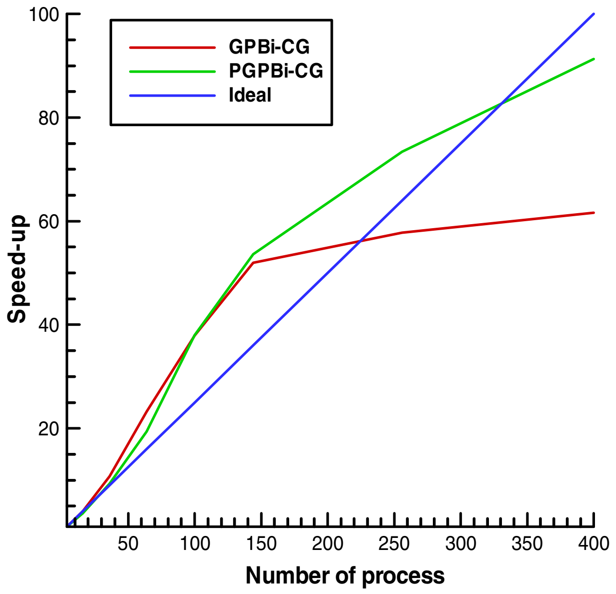

Fig. 3 Speed-up based on 1 processor

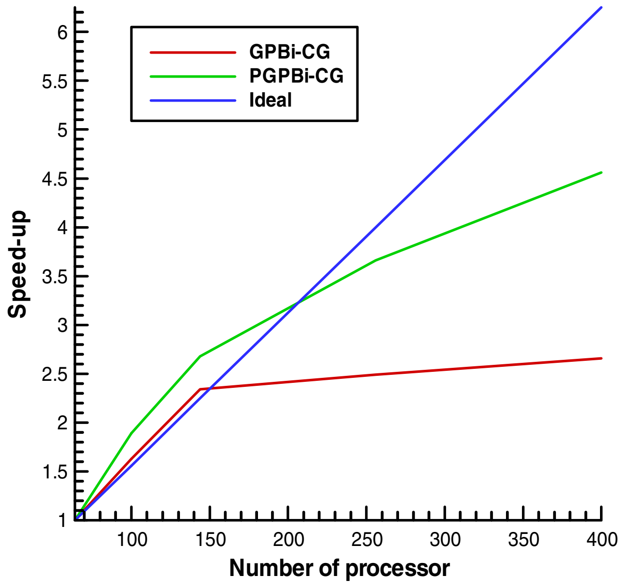

Fig. 4 Speed-up based on 64 processors

ance for convergence is residual 2-norm less than 10^{-6}. We find that as the number of CPUs becomes larger, the number of iterations for GPBi-CG becomes larger and larger, while it becomes slightly smaller for PGPBi-CG. The results show that the PGPBi-CG method convergent faster than GPBi-CG. The residual 2-norm of GPBi-CG appears irregular, which increases at the beginning of iteration (Fig. 5) for more processors, stagnation

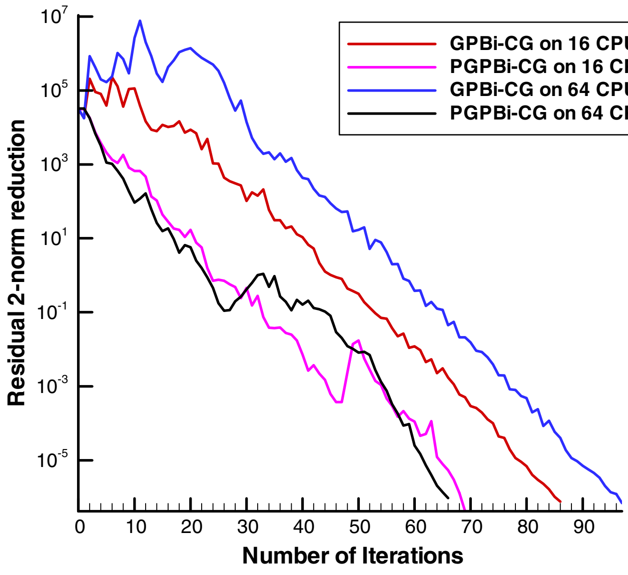

Fig. 5 Comparison of PGPBi-CG and GPBi-CG 16 and 64 processors

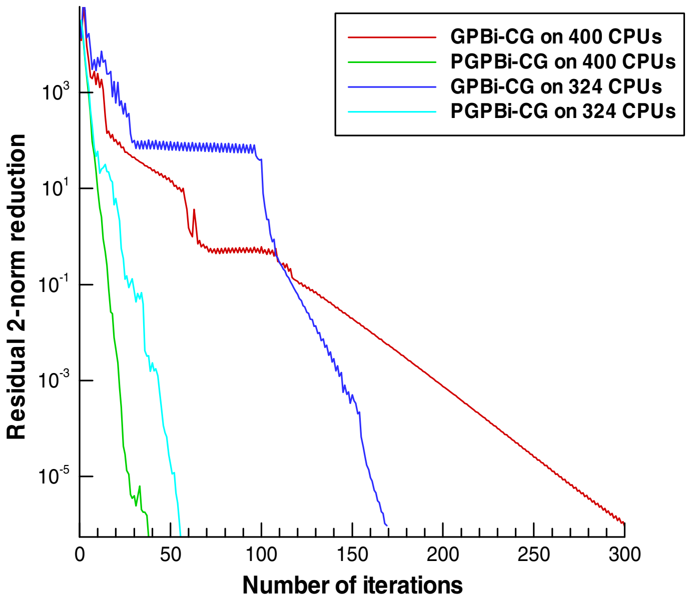

Fig. 6 Comparison of PGPBi-CG and GPBi-CG 324 and 400 processors

occurs (Fig. 6) just like the behavior of GMRES. Compared with GPBi-CG method, the number of iterations for convergence of PGPBi-CG method can improve by a factor up to 6.

6 Conclusion

In this paper, a parallel version of GPBi-CG method for solving large sparse linear systems with unsymmetrical coefficient matrices is proposed for distributed parallel architectures. It combines the elements of numerical stability with the characters of design of parallel algorithms. The cost is only slightly increased computation, which is negligible relative to the improvement in communication performance. Performance and isoefficiency analysis show that the PGPBi-CG method has better parallelism and scalability than GPBi-CG method and the parallel communication performance can be improved by a factor 3. Numerical experiments agree with our analysis. Furthermore, the PGPBi-CG method has better parallel speedup and faster convergence rate compared with the GPBi-CG method.

For further performance improvements, one can consider overlap of computation with communication. Another problem 1 is to improve the numerical stability of the improved method.

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

References

Bhaya A, Bliman PA, Niedu G, Pazos F (2012) A cooperative conjugate gradient method for linear systems permitting multithread implementation of low complexity. In: Proceedings of the IEEE Conference on Decision and Control, pp. 638ff

Bücker HM, Sauren M (1996) 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 TP, vanGijzen MB (2010) 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

de Sturler E, van der Vorst HA (1995) Reducing the effect of the global communication in GMRES(m) and CG on parallel distributed memory computers. Appl Numer Math 18:441–459

de Sturler E (1996) A performance model for Krylov subspace methods on mesh-based parallel computers. Parallel Comput 22:57–74

Freund RW, Nachtigal NM (1991) QMR: a quasi-minimal residual method for non-Hermitian linear systems. Numer Math 60:315–339

Grigori L, Moufawad S (2013) Communication avoiding ILU0 preconditioner. http://hal.inria.fr/docs/00/80/32/50/PDF/RR-8266

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

Gu T-X, Liu X-P, Mo Z-Y (2004) Multiple search direction conjugate gradient method II: theory and numerical experiments. Int J Comput Math 81(10):1289–1307

Gu T-X, Zuo X-Y, Zhang L-T, Zhang W-Q, Sheng Z-Q (2007) An improved bi-conjugate residual algorithm suitable for distributed parallel computing. Appl Math Comput 186:1243–1253

Gu T-X Zuo X-Y, Liu X-P, Li P-L (2009) An improved parallel hybrid bi-conjugate gradient method suitable for distributed parallel computing. J Comput Appl Math 226:56–65

Liu J, Liu X-P, Chi L-H, Hu Q-F, Li X-M (2005) An improved conjugate residual algorithm for large symmetric linear systems. In: Proceedings of the Joint Conference of ICCP6 and CCP2003, Comput. Phys., Xi-Jun Yu (ed.) Rinton Press, New Jersey, USA, pp. 325C328

Liu X-P, Gu T-X, Hang X-D, Sheng Z-Q (2006) A parallel version of QMRCGSTAB method for large linear systems in distributed parallel environments. Appl Math Comput 172(2):744C752

Saad Y (1996) Iterative methods for sparse linear systems. PWS Publishing Company, Boston

Sonneveld P, van Gijzen M (2008) IDR(s): a family of simple and fast algorithms for solving large nonsymmetric systems of linear equations. SIAM J Sci Comput 31(2):1035C1062

Sogabe T, Zhang S-L (2003) Extended conjugate residual methods for solving nonsymmetric linear systems. In: Yuan Y-X (ed) Numerical linear algebra and optimization. Science Press, Beijing/New York, pp. 88–99

van der Vorst HA (1992) 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

Yang TR, Lin HX (1997) 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)

Yang TR (2002) The improved CGS method for large and sparse linear systems on bulk synchronous parallel architectures. In: In 5th International Conference on Algorithms and Architectures for Parallel Processing, IEEE Computer Society, pp. 232–237

Yang TR, Brent RP (2002) The improved BICGSTAB method for large and sparse unsymmetric linear systems on parallel distributed memory architectures. In: 5th International Conference on Algorithms and Architectures for Parallel Processing, IEEE Computer Society, pp. 324–328

Yang TR, Brent RP (2003) The improved BiCG method for large and sparse linear systems on parallel distributed memory architectures. Inf J 6:349–360

Zhang S-L (1997) GPBi-CG: generalized product-type methods based on Bi-CG for solving nonsymmetric linear systems. SIAM J Sci Comput 18:537–551