The Improved Parallel BiCG Method for Large and Sparse Unsymmetric Linear Systems on Distributed Memory Architectures*

Laurence Tianruo Yang†‡, Richard P. Brent ‡
†Department of Computer Science, St. Francis Xavier University
P.O. Box 5000, Antigonish, B2G 2W5, NS, Canada
‡Computing Laboratory, Oxford University
Wolfson Building, Park Road, Oxford OX1 3QD, UK

Abstract

For the solutions of large and sparse linear systems of equations with unsymmetric coefficient matrices, we propose an improved version of the BiConjugate Gradient method (IBiCG) method by using the Lanczos process as a major component combining elements of numerical stability and parallel algorithm design. For the Lanczos process, stability is obtained by a coupled two-term procedure that generates Lanczos vectors scaled to unit length. The algorithm is derived such that all inner products, matrix-vector multiplications and vector updates of a single iteration step are independent and communication time required for inner products can be overlapped efficiently with computation time of vector updates. Therefore, the cost of global communication on parallel distributed memory computers can be significantly reduced. The resulting IBiCG algorithm maintains the favorable properties of the Lanczos process while not increasing computational costs. Data distribution suitable for both irregularly and regularly structured matrices based on the analysis of the nonzero matrix elements is presented. A communication scheme is supported by overlapping execution of computation and communication to reduce waiting times. The efficiency of this method is demonstrated by numerical experimental results carried out on a massively parallel distributed memory system.

Keywords : biconjugate gradient method, large and sparse linear unsymmetric linear systems, distributed memory architectures, data distribution and communication

1 Introduction

One of the fundamental tasks of numerical computing is the ability to solve linear systems. These systems arise very frequently in scientific and engineering computing, for example from finite difference or finite element approximations

*The author's email is lyang@stfx.ca

to partial differential equations, as intermediate steps in computing the solution of nonlinear problems or as subproblems in linear and nonlinear programming.

For linear systems of small size, the standard approach is to use direct methods, such as LU decomposition which obtains the solution through a factorization of the coefficient matrix. In contrast with direct methods, iterative methods use successive approximations to obtain more accurate solutions to the linear systems at subsequent steps. Iterative methods are computationally more attractive than the direct methods particularly for large and sparse systems [11].

A powerful iterative method for the solution of large and sparse linear systems with unsymmetric positive definite coefficient matrices is the family of Krylov subspace methods [10, 14] involving the coefficient matrix only in the form of matrix-by-vector products. These methods involve the generation of a suitable basis of a vector space called Krylov subspace and the choice of the actual iterates within that space. In this paper, we will focus on one of the Krylov subspace methods, namely, the BiConjugate Gradient algorithm [10, 14], for large and sparse linear systems with unsymmetric coefficient matrices.

On massively parallel computers, the basic time-consuming computational kernels of Krylov subspace methods are usually: inner products, vector updates, 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 [9]. 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 not avoid global communication, i.e. accumulation of data from all to one processor, required for inner product computations. Vector updates are perfectly parallelizable 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. The detailed discussions on the communication problem on distributed memory systems can be found in [7, 8, 16]. 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 [7, 8].

Recently, Bücker et al. [5, 6] propose a new modified parallel version of the BiConjugate Gradient (MBiCG) method. The algorithm is derived so that both generated sequences of Lanczos vectors are scalable and is reorganized without changing the numerical stability so that all inner products and matrix-vector multiplications of a single iteration step are independent. There is only a single global synchronization point per iteration. Therefore, the cost of global communication on parallel distributed memory computers can be reduced. Based on similar ideas, we propose a new improved two-term recurrence Lanczos process without look-ahead as the underlying process for the new Improved BiConjugate Gradient (IBiCG) method. The algorithm is reorganized without changing the numerical stability so that all inner products, matrix-vector multiplications and vector updates of a single iteration step are independent, and communication time required for inner products can be overlapped efficiently with computation time of vector updates. Therefore, the cost of global communication on parallel distributed memory computers can be significantly reduced. The resulting IBiCG algorithm maintains the favorable properties of the Lanczos process while not increasing computational costs. The efficient parallel implementation details, in particularly, data distribution and communication scheme, will be addressed as well. The efficiency of this method is demonstrated by numerical experimental results carried out on a massively parallel distributed memory computer, the Parsytec system.

The paper is organized as follows. In section 2, we describe briefly the classical unsymmetric Lanczos algorithm. A sketch of a new improved variant, used as the underlying process for the Improved BiConjugate Gradient (IBiCG) method is given in section 3. In section 4, the parallel implementation details including data distribution and communication scheme are presented. Finally numerical experiments carried out on parallel distributed memory computers are reported, and some comparisons between the new Improved BiConjugate Gradient (IBiCG) method and other approaches are given with regards to numerical accuracy, parallel performance and efficiency.

2 Lanczos Process Based on Coupled Two-Term Recurrence

Although Lanczos used a coupled two-term recurrence in the early of 1950's, the majority of papers deal with the three-term recurrence process, until, recently, Freund et al. [13] reused the old idea to improve numerical stability. They showed that, the latter variant of the Lanczos process, is numerically more stable. That is why we pursue further this unsymmetric Lanczos process with two-term recurrences as the underlying process of the BiCG method.

Recently, Bücker et al. [3, 4] propose a new parallel version of the quasi-minimal residual (QMR) method based on the coupled two-term recurrence Lanczos process without look-ahead strategy. The algorithm is derived so that both generated sequences of Lanczos vectors are scaled to unit length and there is only one single global synchronization point per iteration. Based on similar ideas, we will present a new improved coupled two-term recurrences Lanczos process without look-ahead technique.

Here we assume that the tridiagonal matrix T has an LU decomposition as

$$T = LU, \tag{1}$$

where the factors L and U are of lower and upper bidiagonal form, respectively. It is the bidiagonal structure of L and U that results in coupled two-term recurrences.

It has been pointed out by several authors [11, 12, 15, 17] that in practice we should scale both sequences of Lanczos vectors appropriately to unit length in order to avoid over- and underflow. This can only be achieved by giving up the bidiagonality $W^T V = I$ and setting $W^T V = D$ instead, where D is a diagonal matrix with diagonal entries $\delta_i \neq 0$ for $i = 1, 2, \cdots, n$.

The principal idea of the new approach suggested in [3, 4] is to start from the scaling described above by using the LU decomposition as well as introducing $P = VU^{-1}$ and $\tilde{Q} = WD^{-1}U^T$, which leads to

$$W^T V = D, \qquad V = PU, \tag{2}$$

and

$$\tilde{Q} = WD^{-1}U^T, \quad AP = VL, \quad A^T W = \tilde{Q}L^T D. \tag{3}$$

Assume that the matrices introduced by the above setting have column vectors according to

$$P = [p_1, p_2, \cdots, p_n] \quad \text{and} \quad \tilde{Q} = [\tilde{q}_1, \tilde{q}_2, \cdots, \tilde{q}_n].$$

Then, omitting details of the derivation, we can get

$$\begin{aligned} p_n &= \frac{1}{\gamma_n}\tilde{v}_n - \mu_n p_{n-1}, \quad \tilde{v}_{n+1} = u_n - \frac{\tau_n}{\gamma_n}\tilde{v}_n, \\ u_n &= \frac{1}{\gamma_n}A\tilde{v}_n - \mu_n u_{n-1}, \quad \tilde{w}_{n+1} = q_n - \frac{\tau_n}{\xi_n}\tilde{w}_n, \\ q_n &= \frac{1}{\xi_n}A^T\tilde{w}_n - \frac{\gamma_n\mu_n}{\xi_n}q_{n-1}. \end{aligned}$$

and the corresponding coefficients are given as follows

$$\begin{aligned} \gamma_{n+1} = (\tilde{v}_{n+1}, \tilde{v}_{n+1}), &\quad \xi_{n+1} = (\tilde{w}_{n+1}, \tilde{w}_{n+1}), \\ \rho_{n+1} = (\tilde{w}_{n+1}, \tilde{v}_{n+1}), &\quad \varepsilon_{n+1} = (A^T\tilde{w}_{n+1}, \tilde{v}_{n+1}), \\ \mu_{n+1} = \frac{\gamma_n\xi_n\rho_{n+1}}{\gamma_{n+1}\tau_n\rho_n}, &\quad \tau_{n+1} = \frac{\varepsilon_{n+1}}{\rho_{n+1}} - \gamma_{n+1}\mu_{n+1}. \end{aligned}$$

Algorithm 1 Improved Lanczos Process

1: $p_0 = q_0 = u_0 = 0, \gamma_1 = (\tilde{v}_1, \tilde{v}_1), \xi_1 = (\tilde{w}_1, \tilde{w}_1), s_1 = A^T\tilde{w}_1,$
2: $\rho_1 = (\tilde{w}_1, \tilde{v}_1), \varepsilon_1 = (s_1, \tilde{v}_1), \mu_1 = 0, \tau_1 = \frac{\varepsilon_1}{\rho_1};$
3: **for** n= 1,2,... **do**
4: $\quad q_n = \frac{1}{\xi_n}s_n - \frac{\gamma_n\mu_n}{\xi_n}q_{n-1};$
5: $\quad \tilde{w}_{n+1} = q_n - \frac{\tau_n}{\xi_n}\tilde{w}_n;$
6: $\quad s_{n+1} = A^T\tilde{w}_{n+1};$
7: $\quad t_n = A\tilde{v}_n;$
8: $\quad u_n = \frac{1}{\gamma_n}t_n - \mu_n u_{n-1};$
9: $\quad \tilde{v}_{n+1} = u_n - \frac{\tau_n}{\gamma_n}\tilde{v}_n;$
10: $\quad p_n = \frac{1}{\gamma_n}\tilde{v}_n - \mu_n p_{n-1};$
11: $\quad \gamma_{n+1}^2 = (\tilde{v}_{n+1}, \tilde{v}_{n+1});$
12: $\quad \xi_{n+1}^2 = (\tilde{w}_{n+1}, \tilde{w}_{n+1});$
13: $\quad \rho_{n+1} = (\tilde{w}_{n+1}, \tilde{v}_{n+1});$
14: $\quad \varepsilon_{n+1} = (s_{n+1}, \tilde{v}_{n+1});$
15: $\quad \mu_{n+1} = \frac{\gamma_n\xi_n\rho_{n+1}}{\gamma_{n+1}\tau_n\rho_n};$
16: $\quad \tau_{n+1} = \frac{\varepsilon_{n+1}}{\rho_{n+1}} - \gamma_{n+1}\mu_{n+1};$
17: **end for**

Now we are able to (re)schedule the operations of in the Lanczos process, in such a way that numerical stability is maintained and all inner products and matrix-vector multiplications of a single iteration step are independent, and communication time required for inner products can be overlapped efficiently with computation time of vector updates. The framework of this improved Lanczos process based on two-term recurrences is described above. In the following, we consider the parallelism of the operations in a single iteration step implemented as follows:

- The inner products of a single iteration step (11), (12), (13) and (14) are independent.

- The matrix-vector multiplications of a single iteration step (6) and (7) are independent.
- The communications required for the inner products (11), (12), (13) and (14) can be overlapped with the update for p_n in (10).

Therefore, the cost of communication time on parallel distributed memory computers can be significantly reduced.

The biorthogonality relationships (2) and (3) are used to derive the algorithm. We can show that, in exact arithmetic, the vectors $\tilde{v}_i$ and $\tilde{w}_i$ generated by the above algorithm are still biorthogonal.

Theorem 2.1 *Assuming that no breakdown occurs, the vectors $\tilde{v}_i$ and $\tilde{w}_i$ generated by the Improved Lanczos Process satisfy*

$$\tilde{w}_i^T \tilde{v}_j = \begin{cases} 0 & i \neq j, \\ \rho_i \neq 0 & i = j. \end{cases}$$

3 The Improved BiConjugate Gradient Method

The following derivation is mainly based on [5, 6]. The improved Lanczos process now is used as a major component to a Krylov subspace method for solving a system of linear equations

$$Ax = b, \quad \text{where} \quad A \in \Re^{n \times n} \quad x, b \in \Re^n. \tag{4}$$

In each step, it produces approximation x_n to the exact solution of the form

$$x_n = x_0 + \mathcal{K}_n(r_0, A), \quad n = 1, 2, ... \tag{5}$$

Here x_0 is any initial guess for the solution of linear systems, $r_0 = b - Ax_0$ is the initial residual, and $\mathcal{K}_n(r_0, A) = span\{r_0, Ar_0, \ldots, A^{n-1}r_0\}$, is the n-th Krylov subspace with respect to r_0 and A.

Given any initial guess x_0, the n-th iterate is of the form

$$x_n = x_0 + V_n z_n, \tag{6}$$

where V_n is generated by the improved unsymmetric Lanczos process, and z_n is determined by the property described later.

For the improved Lanczos process, the n-th iteration step generates

$$V_{n+1} = [v_1, v_2, \cdots, v_{n+1}] \text{ and } P_n = [p_1, p_2, \cdots, p_n],$$

that are connected by

$$P_n = V_n U_n^{-1}, \qquad AP_n = V_{n+1} L_n, \tag{7}$$

where L_n and U_n are the leading principal $(n+1) \times n$ and $n \times n$ submatrices of the bidiagonal matrices L and U generated by the improved Lanczos process. Note that L_n has full rank since we assume no breakdown occurs. The setting of $y_n = U_n z_n$ can be used to reformulate the iterate in term of y_n instead of z_n giving

$$x_n = x_0 + P_n y_n. \tag{8}$$

The corresponding residual vector in term of y_n is obtained by the above scenario, namely

$$\begin{aligned} r_n &= b - Ax_n = r_0 - V_{n+1}L_n y_n \\ &= V_{n+1}(\gamma_1 e_1^{(n+1)} - L_n y_n), \end{aligned} \tag{9}$$

where the improved Lanczos process starts with $v_1 = \frac{1}{\|r_0\|_2} r_0$ and $e_1^{(n+1)} = (1, 0, \ldots, 0)^T$. By using the Lanczos process to generate the Krylov subspace and fixing y_n, and so implicitly z_n by $y_n = U_n z_n$, we can easily derive the iterative method. Let $\widehat{y}_i$ denote the ith component of y_n, i.e., $y_n = (\widehat{y}_1, \widehat{y}_2, \ldots, \widehat{y}_n)^T$. We can derive r_n to the zero vector by

$$\widehat{y}_i = \begin{cases} \frac{\gamma_1}{\tau_1} & \text{if } i = 1, \\ -\frac{\gamma_i}{\tau_i}\widehat{y}_{i-1} & \text{if } i = 2, 3, \ldots, n. \end{cases} \tag{10}$$

This choice of y_n is observed by the special structure of L_n given in [5] and zeros out the first n components of the vector $\gamma_1 e_1^{(n+1)} - L_n y_n$ in (9). The residual vector is then expressed by

$$r_n = V_{n+1} \begin{pmatrix} 0 \\ \vdots \\ 0 \\ -\gamma_{n+1}\widehat{y}_n \end{pmatrix} \tag{11}$$

We can observe that the process of fixing y_n is easily updated in each iteration step because y_{n-1} coincides with the first $n-1$ components of y_n. The corresponding recursion can be described as follows:

$$y_n = \begin{pmatrix} y_{n-1} \\ 0 \end{pmatrix} + \kappa_n e_n \ , \tag{12}$$

where $e_n = (0, \ldots, 0, 1)^T \in \Re^n$ and

$$\kappa_n = -\frac{\gamma_n}{\tau_n}\kappa_{n-1} \tag{13}$$

with $\kappa_0 = -1$. By substituting (12) into (8), it yields

$$x_n = x_0 + P_{n-1}y_{n-1} + \kappa_n p_n \tag{14}$$

$$= x_{n-1} + \kappa_n p_n \ . \tag{15}$$

Note that $\|r_n\|$ is immediately available because, at every iteration step, the last component of y_n is κ_n by (12). Hence, from (11) the following relation can be derived

$$\|r_n\| = \|v_{n+1}\| \cdot |\gamma_{n+1}\kappa_n|. \tag{16}$$

By properly reorganizing (13) and (15) into the improved Lanczos process characterized by equations (6), we can derive results described in Algorithm 2. Note that the underlying Lanczos process operates with unit scaling of both sequences of Lanczos vectors, and is explicitly presented in [2]. In this case, (16) simplifies to

$$\|r_n\| = |\gamma_{n+1}\kappa_n|. \tag{17}$$

We can use it as a simple stopping criterion.

Looking carefully at the above iterative process for the solution of linear systems, we can see that Algorithm 2 is just a new variant of the BiConjugate Gradient method (BiCG) [10, 14]. The detailed description and corresponding discussion can be found in [5, 6].

Algorithm 2 Improved Biconjugate Gradient Method

1: $p_0 = q_0 = 0\,, \widetilde{v}_1 = \widetilde{w}_1 = b - Ax_0$
2: $\gamma_0 = \xi_0 = 0\,, \tau_0 = \rho_0 \neq 0\,, \kappa_0 = -1$
3: $a_0 = A^T q_0, b_0 = Ap_0, s_0 = A^T \widetilde{w}_1$
4: $\gamma_1 = \|\widetilde{v}_1\|, \xi_1 = \|\widetilde{w}_1\|\,, \varrho_1 = \widetilde{w}_1^T v_1\,, \varepsilon_1 = s_1^T \widetilde{v}_1$
5: **for** n = 1, 2, 3, . . . **do**
6: $\quad \mu_n = \frac{\gamma_{n-1}\xi_{n-1}\varrho_n}{\gamma_n \tau_{n-1}\varrho_{n-1}}$
7: $\quad \tau_n = \frac{\varepsilon_n}{\varrho_n} - \gamma_n \mu_n$
8: $\quad \kappa_n = -\frac{\gamma_n}{\tau_n}\kappa_{n-1}$
9: $\quad p_n = \frac{1}{\gamma_n}\widetilde{v}_n - \mu_n p_{n-1}$
10: $\quad q_n = \frac{1}{\xi_n} s_n - \frac{\gamma_n \mu_n}{\xi_n} q_{n-1}$
11: $\quad a_n = A^T q_n$
12: $\quad b_n = Ap_n$
13: $\quad s_{n+1} = a_n - \frac{\tau_n}{\xi_n} s_n$
14: $\quad \widetilde{v}_{n+1} = b_n - \frac{\tau_n}{\gamma_n}\widetilde{v}_n$
15: $\quad \widetilde{w}_{n+1} = q_n - \frac{\tau_n}{\xi_n}\widetilde{w}_n$
16: $\quad \gamma_{n+1} = \widetilde{v}_{n+1}^T \widetilde{v}_{n+1}$
17: $\quad \xi_{n+1} = \widetilde{w}_{n+1}^T \widetilde{w}_{n+1}$
18: $\quad \varrho_{n+1} = \widetilde{w}_{n+1}^T \widetilde{v}_{n+1}$
19: $\quad \varepsilon_{n+1} = s_n^T \widetilde{v}_{n+1}$
20: $\quad x_n = x_{n-1} + \kappa_n p_n$
21: $\quad$ **if** $(|\gamma_{n+1}\kappa_n| < tol)$ **then**
22: $\quad\quad$ STOP
23: $\quad$ **end if**
24: **end for**

Under the assumptions, the Improved BiConjugate gradient (IBiCG) method using improved Lanczos process as underlying process can be efficiently parallelized as follows:

- The inner products of a single iteration step (16), (17), (18), and (19) are independent.
- The matrix-vector multiplications of a single iteration step (11) and (12) are independent.
- The vector updates (13), (14) and (15) are independent.
- The vector updates (9) and (10) are independent.
- The communications required for the inner products (16), (17), (18) and (19) can be overlapped with the update for p_n in (20).

Therefore, the cost of communication time on parallel distributed memory computers can be significantly reduced.

4 Parallel Implementation

4.1 Data distribution

For large and sparse matrices, if we are working on different computer system architectures or dealing with different algorithms or data, appropriately efficient storage schemes should be considered. In this paper, we use one of the most common formats called CRS format (compressed row storage). The main reason is that this type of storage scheme is very suitable for both regularly and irregularly structured large and sparse matrices. The detailed description can be found in the literature. Briefly, the non-zeros of a large and sparse matrix are stored in row-wise in three one-dimensional arrays. The values of the non-zeros are contained in array *value*. The corresponding column indices are contained in array *col_ind*. The elements of *row_ptr* point to the position of the beginning of each row in *value* and *col_ind*.

In order to efficiently parallelize the IBiCG algorithm on a distributed memory architecture, we first need to decide the data distribution of matrix and vector arrays to each processor, and then determine an efficient communication scheme by taking into account different sparsity patterns, not only for matrix-vector multiplication but also for inner products, to minimize the overall execution time. In this paper, we will mainly follow the approach has been used in [1] for data distribution and communication scheme which do not require any knowledge of the matrix sparsity pattern. The communication scheme is automatically determined by the analysis of the indices of the non-zero matrix elements.

In the following part, we will use the same notations introduced in [1] for illustration. Let n_k and e_k denote the number of rows and no-zeros of processor k, where $k = 0, \ldots, p-1$, respectively. e and n are the total number of corresponding numbers. g_k is the index of the first row of processor k, and z_i is the number of non-zeros of row i. We can get the following relations from [1]: $n = \sum_{k=0}^{p-1} n_k, e = \sum_{k=0}^{p-1} e_k, g_k = 1 + \sum_{i=0}^{k-1} n_i, e_k(g_k, n_k) = \sum_{i=gk}^{gk+nk-1} z_i$ Based on the analysis, the total costs of each iteration can be described as $c_1\ s\ e + c_2\ n + c_3$ where the first term corresponds to the number of operations for s matrix-vector multiplications, the second term corresponds to the number of vector updates. Since we are mainly dealing with large and sparse matrices, the constants can be neglected. Now we can estimate the contribution of the operations executed on processor k to the total number of operations by

$$\zeta \approx \frac{c_1\ s\ e_k + c_2\ n_k}{c_1\ s\ e + c_2\ n} = \frac{s\ e_k + \xi\ n_k}{s\ e + \xi n},$$

where $\xi = c_2/c_1$ depends on both the iterative methods and also the processor architecture. Ideally, the computational load balance should be distributed in such a way that each processor only gets p-th fraction for the total number of operations. Based on this, we can use the following strategy to distribute the rows of the matrix and the vector components [1]:

$$n_k = \begin{cases} \min\limits_{1 \le t \le n - g_k + 1} \left\{ t | \frac{s\ e_k(t) + \xi\ t}{s\ e + \xi\ n} \ge \frac{1}{p} \right\} & k = 0, 1, \ldots, q \\ n - \sum_{i=0}^{q} n_i & k = q+1 \\ 0 & k = q+2, \ldots, p-1 \end{cases}$$

Since our main target is large and sparse matrices and we assume $p \ll n$, the relation $q = p-1$ or $q+1 = p-1$ always holds. It can be shown that for $\xi = c_2/c_1 \to 0$, each processor will get nearly the same number of non-zeros which means that the execution time of the vector updates in negligible compared to the execution of matrix-vector multiplications. It also can be shown that for $\xi = c_2/c_1 \to \infty$ each processor will get nearly the same number of rows which means that the execution time of the matrix-vector multiplications only contribute to a very small part of the total execution time.

4.2 Communication schemes

After the discussion of data distribution, we also need to investigate a suitable communication scheme by preprocessing the distributed column index arrays for efficient matrix-vector multiplications, since on a distributed memory systems, its computation requires communication due to the partial vector on each processor. We will use an approach similar to that proposed in [1] for our communication schemes.

If we decide to implement the matrix vector multiplication row-wisely, components of the vector x of $y = Ax$ are communicated. We firstly analyze the arrays *col_ind* on each processor to determine which elements result in access to non-local data. Then, the processors exchange information to decide which local data must be sent to which processors. Based on the above analysis, we will reorder these two arrays *col_ind* and *value* in such a way that the data that results in access to processor l is collected in block l, called a *local block*. The motivation behind this reordering is to perform computation and communication overlapped. The elements of block l succeed one another row-wise with increasing column index per row. The detailed description can be found in [1].

For the parallel implementation of this operation, each processor executes asynchronous receive-routines to receive necessary non-local data. Then all components of the vector x that are needed on other processors are sent asynchronously. After the required data are available, each processor will perform operations with its local block. After that, as soon as non-local data arrive, the processor continues the matrix vector operation by accessing the elements of the corresponding blocks. This is repeated until the whole operation is completed. According to the communication scheme described, the communication and computation are overlapped, so that waiting time can be reduced.

5 Numerical Experiments

In this section, the parallel variant of the improved BiCG (IBiCG) using the improved Lanczos process as the underlying Lanczos process is compared with the modified version of BiCG (MBiCG) proposed by Bücker et al. [5, 6] and the original BiCG based on coupled two-term recurrences on a massively distributed memory Patsytec computer.

Here we mainly consider the partial differential equation taken from [3, 5, 6]

$$Lu = f, \qquad \text{on} \qquad \Omega = (0,1) \times (0,1),$$

with Dirichlet boundary condition $u = 0$ where

$$Lu = -\Delta u - 20 \left(x \frac{\partial u}{\partial x} + y \frac{\partial u}{\partial y} \right),$$

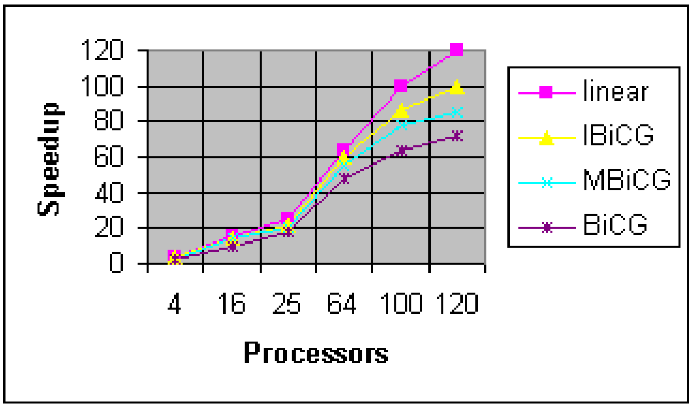

Figure 1: Experimental results of speed-up

and the right-hand side f is chosen so that the solution is

$$u(x,y) = \frac{1}{2}\sin(4\pi x)\sin(6\pi y).$$

Basically, we discretize the above differential equation using second order centered differences on a 400×400 mesh with mesh size $1/441$, leading to a system of 193600 linear equations with an unsymmetric coefficient matrix of 966240 nonzero entries. Diagonal preconditioning is used. For our numerical tests, we choose $x_0 = 0$ as initial guess and $tol = 10^{-5}$ as stopping parameter.

Since the vectors are distributed over the processor grid, the inner products usually are computed in two steps. All processors start to compute in parallel the local inner products. After that, the local inner products are accumulated on one central processor and broadcast. The communication time of an accumulation or a broadcast increases in proportion to the diameter of the processor grid. That means if the number of processors increases then the communication time for the inner products increases as well, and hence this is a potential threat to the scalability of the algorithm.

The convergence of the proposed improved BiCG (IBiCG) is almost the same as that of the modified BiCG (MBiCG) suggested by Bücker et al. [5] and that of the original BiCG version based on two-term recurrences where $\|r_n\|_2$ is computed recursively. A similar numerical behavior to these variants is observed. There is hardly any difference with respect to the true residual norm $\|b - Ax_n\|_2$ in those versions. The parallel performance is given in Figure 1 where "linear" is the theoretical linear speedup, IBiCG is the speedup of the improved BiCG method, MBiCG is the speedup of the modified BiCG method suggested by Bücker et al. [5, 6] and BiCG is the speedup of the original BiCG method with two-term recurrences. These results are based on timing measurements of a fixed number of iterations. Since we do not know exactly the implementation details of the modified BiCG method [5, 6], we only take the published data as the experimental data for comparison. The speedup is computed as the ratio of the parallel execution time and the execution time using one processor. From the results, we can see clearly that the modified BiCG (MBiCG) suggested by Bücker et al. [5, 6] is faster than the original one. Meanwhile the new approach can achieve much better parallel performance

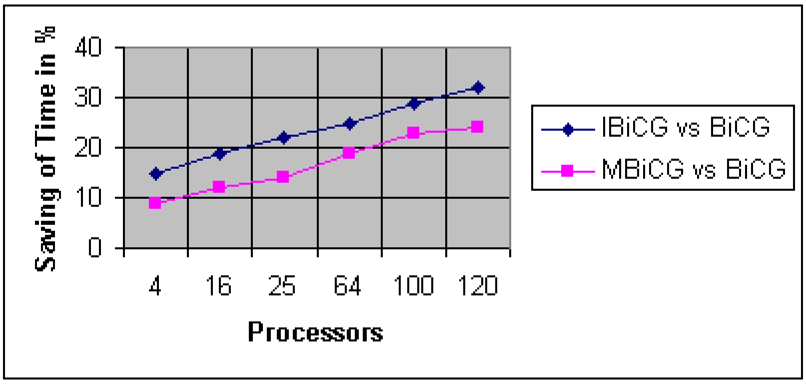

Figure 2: Execution time reduction

with a higher scalability than the modified one. In comparison to the two other approaches, the reduction in execution time by the IBiCG increases with the number of processors. More precisely, the quantity is $1 - T_A(p)/T_B(p)$, where $T_A(p)$ and $T_B(p)$ are the execution times on p processors of approach A and B respectively. In Figure 2, the first curve shows the percentage of reduction in execution time by our improved BiCG (IBiCG) approach compared to the original one. Another curve shows the percentage of reduction of time for the modified BiCG (MBiCG) proposed by Bücker et al. [5, 6] compared to the original BiCG method.

6 Conclusions

We have described an improved version of the BiConjugate Gradient method (IBiCG) method by using the Lanczos process as a major component combining elements of numerical stability and parallel algorithm design for the solution of large and sparse linear systems of equations with unsymmetric coefficient matrices. For the Lanczos process, stability is obtained by a coupled two-term procedure that generates Lanczos vectors scaled to unit length. The algorithm is derived in such a way that all inner products, matrix-vector multiplications and vector updates of a single iteration step are independent and communication time required for inner products can be overlapped efficiently with computation time of vector updates. Therefore, the cost of global communication on parallel distributed memory computers can be significantly reduced. The resulting IBiCG algorithm maintains the favorable properties of the Lanczos process while not increasing computational costs. Data distribution suitable for both irregularly and regularly structured matrices based on the analysis of the non-zero matrix elements has been presented. Communication scheme can be supported by overlapping execution of computation and communication to reduce waiting times. The approach demonstrates a high scalability on a massively parallel distributed memory computer.

References

1. A. Basermann, B. Reichel, and C. Schelthoff. Preconditioned CG methods for sparse matrices on massively parallel machines. *Parallel Computing*, 23(3):381–398, 1997.

2. H. M. Bücker and M. Sauren. A parallel version of the unsymmetric lanczos algorithm and its application to QMR. Internal Report KFA–ZAM–IB–9605, Research Centre Jülich, Jülich, Germany, March 1996.

3. H. M. Bücker and M. Sauren. A parallel version of the quasi-minimal residual method based on coupled two-term recurrences. In J. Waśniewski, J. Dongarra, K. Madsen, and D. Olesen, editors, *Proceedings of Workshop on Applied Parallel Computing in Industrial Problems and Optimization (Para96), LNCS184*, Lecture Notes in Computer Science, pages 157–165. Technical University of Denmark, Lyngby, Denmark, Springer-Verlag, August 1996.

4. H. M. Bücker and M. Sauren. A parallel version of the unsymmetric Lanczos algorithm and its application to QMR. Technical Report KFA-ZAM-IB-9605, Central Institute for Applied Mathematics, Research Centre Julich, Germany, March 1996.

5. H. M. Bücker and M. Sauren. A variant of the biconjugate gradient method suitable for massively parallel computing. In G. Bilardi, A. Ferreira, R. Lüling, and J. Rolim, editors, *Solving Irregularly Structured Problems in Parallel, Proceedings of the Fourth International Symposium, IRREGULAR'97, Paderborn, Germany, June 12–13, 1997*, volume 1253 of *Lecture Notes in Computer Science*, pages 72–79, Berlin, 1997. Springer.

6. H. M. Bücker and M. Sauren. Parallel biconjugate gradient methods for linear systems. In L. T. Yang, editor, *Parallel Numerical Computations with Applications*, number 51-70. Kluwer Academic Publishers, 1999.

7. E. de Sturler. A parallel variant of the GMRES(m). In *Proceedings of the 13th IMACS World Congress on Computational and Applied Mathematics*. IMACS, Criterion Press, 1991.

8. E. de Sturler and H. A. van der Vorst. Reducing the effect of the global communication in GMRES(m) and CG on parallel distributed memory computers. Technical Report 832, Mathematical Institute, University of Utrecht, Utrecht, The Netherland, 1994.

9. J. J. Dongarra, I. S. Duff, D. C. Sorensen, and H. A. van der Vorst. *Solving Linear Systems on Vector and Shared Memory Computers*. SIAM, Philadelphia, PA, 1991.

10. R. Fletcher. Conjugate gradient methods for indefinite systems. In G. A. Watson, editor, *Numerical Analysis Dundee 1975*, volume 506 of *Lecture Notes in Mathematics*, pages 73–89, Berlin, 1976. Springer.

11. R. W. Freund, G. H. Golub, and N. Nachtigal. Iterative solution of linear systems. *Acta Numerica*, pages 57–100, 1991.

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

13. R. W. Freund and N. M. Nachtigal. An implementation of the QMR method based on coupled two-term recurrences. *SIAM Journal on Scientific and Statistical Computing*, 15(2):313–337, 1994.

14. C. Lanczos. Solutions of systems of linear equations by minimized iterations. 49(1):33–53, 1952.

15. B. N. Parlett, D. R. Taylor, and Z. A. Liu. A look-ahead Lanczos algorithm for unsymmetric matrices. *Mathematics of Computation*, 44:105–124, 1985.

16. C. Pommerell. *Solution of large unsymmetric systems of linear equations.* PhD thesis, ETH, 1992.

17. D. R. Taylor. *Analysis of the look ahead Lanczos algorithm for unsymmetric matrices.* PhD thesis, Department of Mathematics, University of California at Berkeley, November 1982.