The Improved Conjugate Gradient Squared (ICGS) Method on Parallel Distributed Memory Architectures *

Laurence Tianruo Yang†‡, *Richard P. Brent* ‡
†*Department of Computer Science, St. Francis Xavier University*
P.O. Box 5000, Antigonish, B2G 2W5, Nova Scotia, 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 Conjugate Gradient Squared method (ICGS) method. 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 product 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 ICGS algorithm maintains the favorable properties of the algorithm 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 is also presented. 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.

1 Introduction

One of the fundamental task 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 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 [7].

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 [6, 8] involving the coefficient matrix only in the form of matrix-by-vector products. These methods basically consist of the generation of a suitable basis of a vector space called Krylov subspace and the choice of the actual iterate within that space. In this paper, we will mainly focus on one of the Krylov subspace methods, namely, the Conjugate Gradient Squared algorithm [11], 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 [5]. 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 the 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 between only nearby processors. The bottleneck is usually due to inner products enforcing global communi-

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

1530-2016/01 $10.00 © 2001 IEEE

cation. The detailed discussions on the communication problem on distributed memory systems can be found in [3, 4, 10]. 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 [3, 4].

Recently, Maheswaran et al.[9] propose a new modified parallel version of the Conjugate Gradient Squared (MCGS) method, where the synchronization overhead is effectively reduced by a factor of two. This is achieved by changing the computation sequence in the CGS algorithm. Based on their similar ideas, we propose a new Improved Conjugate Gradient Squared (CGS) 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 subsequently communication time required for inner product 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 CGS algorithm maintains the favorable properties of the CGS algorithm 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 will describe briefly the classical Conjugate Gradient Squared (CGS) algorithm. A sketch of a new improved variant, the Improved Conjugate Gradient Squared (ICGS) method is described fully as well. In section 3, the parallel implementation details including data distribution and communication scheme are presented. Numerical experiments carried out on parallel distributed memory computers are reported and some comparisons between the new Improved Conjugate Gradient Squared (ICGS) method and other approaches are given with regards to numerical accuracy, parallel performance and efficiency in section 4. Finally we offer some conclusions.

2 The CGS and ICGS Methods

The Conjugate Gradient Squared (CGS) algorithm [11] for the solution of the linear equations

$$Ax = b, \quad \text{where} \quad A \in \Re^{n\times n} \quad x, b \in \Re^n. \qquad (1)$$

is described in the Algorithm 1. Here x_0 is any initial guess for the solution and $r_0 = b - Ax_0$ is the initial residual. An arbitrarily chosen vectors $\tilde{r}_0$ such that $\tilde{r}_0^T r_0 \neq 0$ are input to the algorithm, in addition to the vector b and the sparse matrix A.

Algorithm 1 The Classical CGS Method

$r_0 = b - Ax_0; q_0 = p_{-1} = 0;$
$\rho_{-1} = 1; \rho_0 = \tilde{r}_0^T r_0;$
for n = 1, 2, 3, ... **do**
  $\beta = \frac{\rho_n}{\rho_{n-1}};$
  $u_n = r_n + \beta q_n;$
  $p_n = u_n + \beta(q_n + \beta p_{n-1});$
  $v_n = Ap_n;$
  $\sigma = \tilde{r}_0^T v_n;$
  $\alpha = \frac{\rho_n}{\sigma};$
  $q_{n+1} = u_n - \alpha v_n;$
  $f_{n+1} = u_n + q_{n+1};$
  $r_{n+1} = r_n - \alpha A f_{n+1};$
  $x_{n+1} = x_n + \alpha f_{n+1};$
  $\rho_{n+1} = \tilde{r}_0^T r_{n+1};$
  $e_{n+1} = r_{n+1}^T r_{n+1};$
  if $(|e_{n+1}| < tol)$ **then**
    STOP
  end if
end for

Recently, Maheswaran et al.[9] propose a new modified parallel version of the Conjugate Gradient Squared (MCGS) method, where the synchronization overhead is effectively reduced by a factor of two. This is achieved by changing the computation sequence in the CGS algorithm. Based on their similar ideas, we propose a new Improved Conjugate Gradient Squared (CGS) method. The basic idea in reformulating the ICGS algorithm is to merge the main computational kernels such as vector updates, matrix-vector multiplications and inner products present in the CGS algorithm so that they can be executed in parallel per iteration of the algorithm. If we reorganize the vectors w_{n+1}, x_{n+1} and r_{n+1} of the MCGS algorithm [9], we can get the following for the vector w_{n+1}:

$$w_{n+1} = Aq_{n+1} = A(u_n - \alpha y_n). \qquad (2)$$

If we define $s_n = Ar_n$ and $t_n = Ay_n$, we can get from (2):

$$w_{n+1} = s_n + \beta w_n - \alpha t_n. \qquad (3)$$

Similarly for the vector x_{n+1}, if we insert $q_{n+1} = u_n - \alpha y_n$ into the expression, we will have

$$\begin{aligned} x_{n+1} &= x_n + \alpha(u_n + q_{n+1}) \\ &= x_n + \alpha(2u_n - \alpha y_n), \end{aligned} \qquad (4)$$

and for the vector r_{n+1}, if we combine w_{n+1} in equation (3), we have

$$r_{n+1} = r_n - \alpha(s_n + \beta w_n + w_{n+1})$$

$$= r_n - \alpha(2s_n + 2\beta w_n - \alpha t_n). \quad (5)$$

Based on the equations (3), (4), (5) and complicated mathematical derivations, the ICGS algorithm can be depicted as follows:

Algorithm 2 The Improved CGS Method

1: $\gamma_{-1} = 1; p_0 = 0; q_0 = 0; w_0 = 0;$
2: $w_{-1} = 0; y_{-1} = 0; r_0 = b - Ax_0; s_0 = Ar_0;$
3: $\gamma_0 = \tilde{r}_0^T r_0; \eta_0 = \tilde{r}_0^T s_0;$
4: $\mu_0 = \tilde{r}_0^T w_0; \lambda = \tilde{r}_0^T y_{-1}; \rho_{-1} = 1;$
5: **for** n = 1, 2, 3, ... **do**
6: $\beta_n = \frac{\gamma_n}{\gamma_{n-1}};$
7: $\sigma_n = \eta_n + \beta_n(2\mu_n + \lambda_n\beta_n);$
8: $u_n = r_n + \beta_n q_n;$
9: $y_n = s_n + 2\beta_n w_n + \beta_n^2 y_{n-1};$
10: $s_n = Ar_n;$
11: $t_n = Ay_n;$
12: $w_n = Aq_n;$
13: $w_{n+1} = s_n + \beta_n w_n - \alpha_n t_n;$
14: $r_{n+1} = r_n - \alpha_n(2s_n + 2\beta_n w_n - \alpha_n t_n);$
15: $x_{n+1} = x_n + \alpha_n(2u_n - \alpha_n y_n);$
16: $\rho_{n+1} = \tilde{r}_0^T r_{n+1};$
17: $\eta_{n+1} = \tilde{r}_0^T s_{n+1};$
18: $\mu_{n+1} = \tilde{r}_0^T w_{n+1};$
19: $\lambda_{n+1} = \tilde{r}_0^T y_n;$
20: $e_{n+1} = r_{n+1}^T r_{n+1};$
21: **if** $(|e_{n+1}| < tol)$ **then**
22: STOP
23: **end if**
24: **end for**

Under the assumptions, the Improved Conjugate Gradient Squared (ICGS) method can be efficiently parallelized as follows:

- The inner products of a single iteration step (16), (17), (18), (19) and (20) are independent(parallel).
- The matrix-vector multiplications of a single iteration step (10), (11) and (12) are independent(parallel).
- The vector updates (13), (14) and (15) are independent, (8) and (9) are independent(parallel).
- The communications required for the inner products (16), (17), (18), (19) and (20) can be overlapped with the update for x_{n+1} in (15).

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

3 Parallel Implementation

3.1 Data distribution

For large and sparse matrices, if you are working on different computer system architectures or dealing with different algorithms or data, the efficient storage schemes should be considered differently. In this paper, we decide to use one of the most common format called CRS format (compressed row storage). The main reason behind 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 speaking, the nonzeros of large and sparse matrix are stored in row-wise in three one-dimensional arrays. The values of the nonzeros 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 ICGS algorithm, in particularly, on a distributed memory architecture, we first need to decide the data distribution of matrix and vector arrays, hopefully optimally, 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 about the matrix sparsity pattern. Also the communication scheme are 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 illustrations. 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. Easily we can get the following relations from [1]: $n = \sum_{k=0}^{p-1} n_k, e = \sum_{k=0}^{p-1} e_k, g_k = 1 + \sum_{i=0}^{k-1} n_i, e_k(g_k, n_k) = \sum_{i=g_k}^{g_k+n_k-1} z_i$ 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]:

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

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 hold. It can be shown that for $\xi = c_2/c_1 \to 0$, each processor will get nearly the same number of nonzeros which means that the execution time of the vector updates in negligible with 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.

3.2 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. Similarly we will use the approach 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 *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 it local block. After that, as soon as non-local data arrive, processor continue the matrix vector operation by accessing the elements of the corresponding blocks. It will be repeated until the whole operation is completed. According to the communication scheme described, the communication and computation are performed overlapped so that waiting time can be reduced.

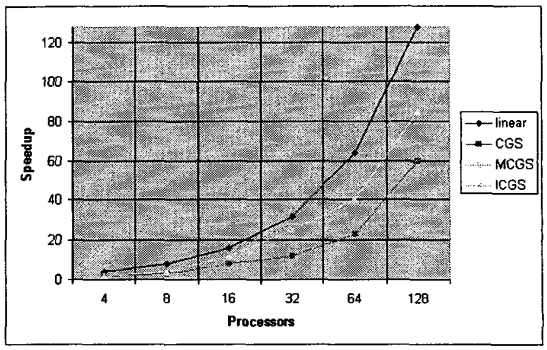

Figure 1. Experimental results of speed-up

4 Numerical Experiments

In this section, the parallel variant of the improved CGS (ICGS) is compared with the modified version of CGS (MCGS) proposed by Maheswaran et al. [9] and the originally classical CGS on a massively distributed memory Parsytec computer.

Here we mainly consider the partial differential equation taken from [2]

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

with Dirichlet boundary condition $u = 0$ where

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

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 with mesh size $1/441$, leading to a system of 193600 linear equations with a 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 broadcasted. The communication time of an accumulation or a broadcast increases proportionally with the diameter of the processor grid. That means if the number of processors increases then the communication time for

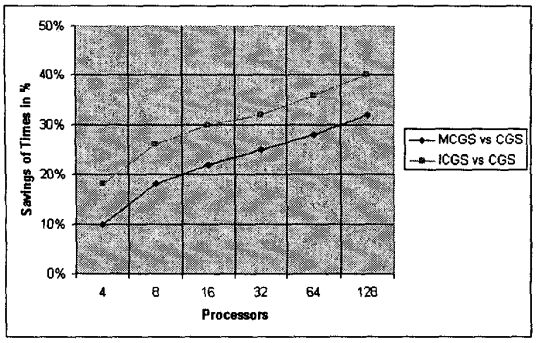

Figure 2. Execution time reduction

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 CGS (ICGS) is almost the same as the modified CGS (MCGS) suggested by Maheswaran et al [9] and original CGS version 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 are given in Fig. 1 where linear is the theoretical linear speedup, ICGS is the speedup of the improved CGS method, MCGS is the speedup of the modified CGS method suggested by Maheswaran et al [9] and CGS is the speedup of the classical CGS method. These results are based on timing measurements of a fixed number of iterations. 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 CGS (MCGS) suggested by Maheswaran et al [9] is faster than the original one. Meanwhile the new approach can achieve much better parallel performance with a higher scalability than the modified one. In comparison to the two other approaches, the reduction in execution time by the ICGS 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 Fig. 2, first curve shows the percentage of reduction in execution time by our improved CGS (ICGS) approach compared to the original one. Another curve shows the percentage of reduction of time for the modified CGS (MCGS) proposed by by Maheswaran et al. [9] compared to the classical CGS method.

5 Conclusions

We have described an improved version of the Conjugate Gradient Squared method (ICGS) method by combining elements of numerical stability and parallel algorithm design for the solution of large and sparse linear systems of equations with unsymmetric coefficient matrices. 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 subsequently communication time required for inner product 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 ICGS algorithm maintains the favorable properties of the algorithm while not increasing computational costs. Data distribution suitable for both irregularly and regularly structured matrices 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):381–398, 1997.
2. 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.
3. 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.
4. 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.
5. 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.
6. 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.
7. R. W. Freund, G. H. Golub, and N. Nachtigal. Iterative solution of linear systems. *Acta Numerica*, pages 57–100, 1991.
8. C. Lanczos. Solutions of Systems of Linear Equations by Minimized Iterations. *Journal of Research of the National Bureau of Standards*, 49(1):33–53, 1952.
9. M. Maheswaran, K. J. Webb, and H. J. Siegel. MCGS: a modified conjugate gradient squared algorithm for nonsymmetric linear systems. *International Journal of Supercomputing*, 2000.
10. C. Pommerell. *Solution of large unsymmetric systems of linear equations*. PhD thesis, ETH, 1992.
11. P. Sonneveld. CGS: a fast Lanczos-type solver for nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 10:36–52, 1989.