A New Version of Conjugate Gradient Method Parallel Implementation

Robert Piotr Bycul
Technical University of Bialystok
Faculty of Electrical Engineering
Grunwaldzka 11/15
15-893 Bialystok, Poland
rpbyc@vela.pb.bialystok.pl

Andrzej Jordan
Technical University of Bialystok
Faculty of Electrical Engineering
Grunwaldzka 11/15
15-893 Bialystok, Poland
jordana@cksr.ac.bialystok.pl

Marcin Cichomski
Polish-Japanese Institute of Information Technology
Koszykowa 86
02-008 Warsaw, Poland
s1270@pjwstk.edu.pl

Abstract

In the article the authors describe an idea of parallel implementation of a conjugate gradient method in a heterogeneous PC cluster and a supercomputer Hitachi SR-2201. The new version of algorithm implementation differs from the one applied earlier[1], because it uses a special method for storing sparse coefficient matrices: only non-zero elements are stored and taken into account during computations, so that the sparsity of the coefficient matrix is taken full advantage of.

The article includes a comparison of the two versions. A speedup of the parallel algorithm has been examined for three different cases of coefficient matrices resulting in solving different physical problems. The authors have also investigated a preconditioning method, which uses the inversed diagonal of the coefficient matrix, as a preconditioning matrix.

1. Introduction

The conjugate gradient method is one of the most popular iterative method and is applied widely, especially for solving sparse, large systems of linear algebraic equations, as well as in optimization problems. However, parallel implementations of the method applied in practice are not universal (suitable for all physical problems) – and they cannot be, because the convergence rate of the method strongly depends on the properties of a coefficient matrix. For symmetric, positive definite matrices, there are a lot of very good serial and parallel implementations, but a problem arises, when the matrix is not symmetric or positive definite. There are a few modifications of the method that help to achieve the convergence in such cases, but the convergence rate is still strongly dependent on the problem structure and usually is much worse than in the unmodified versions. To improve the convergence rate an operation called preconditioning is commonly applied. But the practice again shows, that the convergence rate is then closely attached to the problem-solver-preconditioner combination.

A parallel implementation of the preconditioned conjugate gradient method helps to shorten the computation time and it often even brings a possibility of solving particular, very large problem that could not be solved on a single machine because of its limitations (especially in capacity of a physical memory). One can create his own version, which works optimally for the particular problem structure and in the actual hardware environment. It can be especially useful in a heterogeneous computer cluster. Taking into account the differences in the computational power between the particular computers in a cluster, it can be important to appropriately distribute the computational data.

In the version implemented by the authors the data for computations are distributed unevenly among all the processors. Matrix transposition and matrix-matrix multiplication operations are avoided (comparing to the previous version described in [1]). This allowed the authors to apply a new way of data partitioning, which is much better when it comes to a storage consumption, as well as to a number of computations. The coefficient matrix is partitioned during reading the input data file and the parts of the matrix are immediately sent to the appropriate processors. This makes possible to solve much larger problems than in the previous version.

The article includes a comparison of the two versions. A speedup of the parallel algorithm has been examined with three different cases of coefficient matrices resulting from solving different physical problems. The authors

Proceedings of the International Conference on Parallel Computing in Electrical Engineering (PARELEC'02)
0-7695-1730-7/02 $17.00 © 2002 IEEE

have also investigated a preconditioning method, which uses the inversed diagonal of the coefficient matrix, as a preconditioning matrix [4]. The influence of the preconditioning in the three cases has been examined.

2. Sequential conjugate gradient version

Having a system of linear algebraic equations given in the form:

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

we can write down the sequential conjugate gradient algorithm (SCGA) (on which the parallel implementation was based) as follows [2]:
We start with:

$$x_1 = [0]^T\,,\ r_1 = b - Ax_1,\ p_1 = \frac{A^T r_1}{\left\langle A^T r_1, A^T r_1 \right\rangle} \tag{2}$$

We iterate, for k=2, 3, …:

$$\alpha_k = \frac{1}{\left\langle Ap_k, Ap_k \right\rangle} \tag{3a}$$

$$x_{k+1} = x_k + \alpha_k p_k \tag{3b}$$

$$r_{k+1} = r_k - \alpha_k A p_k \tag{3c}$$

$$\beta_k = \frac{1}{\left\langle A^T r_{k+1}, A^T r_{k+1} \right\rangle} \tag{3d}$$

$$p_{k+1} = p_k + \beta_k A^T r_{k+1} \tag{3e}$$

We finish, when:

$$\frac{\|r_{k+1}\|}{\|b\|} < \varepsilon \tag{4}$$

In the above equations A is a square coefficient matrix, b is a right hand side vector, x is an unknown vector, x_1 is a starting vector of unknowns, r is a residual vector defined at the beginning as in equation (2), p is a search direction vector. α_k and β_k are real numbers. ε is also a real number that determines the accuracy of the computed vector x (i.e. how "far" the vector is from the exact solution). The angle brackets denominate a vector inner product operation and the double "straight" brackets denominate an Euclidean norm of a vector.

The algorithm was taken from [2], so the reader can find there much more information relating to it.

3. Parallel implementation of the SCG

3.1 The concept

The parallel implementation of the SCG algorithm has been constructed basing on already verified [1], [6], [7] assumption that it is necessary to compute only the most time consuming operations (matrix-vector, transposed matrix-vector multiplications and vector inner product) in parallel to get speedup of the computations.

3.2 Data decomposition

There are three operations computed in parallel in the parallel conjugate gradient algorithm (PCGA):
- matrix-vector multiplication,
- (transposed matrix)-vector multiplication,
- vector inner product.

The first two of them are very similar, because the only difference between them is that appropriate indexes of the matrix elements are exchanged to get an effect, as if the matrix was transposed, without actually transposing it in the computer memory.

One can then see that the only matrix that appears in the first two operations is A. So the matrix is distributed among all computational nodes right after it is read from a hard disk by the master process and it stays unchanged during the whole computation time.

Each node has then at its disposal its own part of matrix A and when the need of computing one of the two operations arise, it only gets an appropriate vector from the master process. The vectors are not decomposed, but sent as a whole. This is caused by the fact that the non-zero elements in matrix A are not sorted, so we never know which element from the vector would be needed during multiplying at a specified time.

In the vector inner product operation the particular node gets appropriate parts of both vectors to perform the computation. After all the nodes finish their computations (including the master process), they send their partial results to the master process, which adds them all (including its own result) and this way completes the global result.

Computational data is decomposed unevenly between the nodes following a rule that the faster the processor is, the more computations it should carry. A number of data that particular processor is to compute can be determined by equation (5):

$$D_i = INT\left(\frac{s_i}{\sum_{i=1}^{p} s_i} \cdot n\right). \tag{5}$$

n is a number of all the non-zero elements in the coefficient matrix (in a matrix-vector multiplications) or a number of all elements in a vector (in a vector inner product operation). s_i is a number that is proportional to the speed of particular processor. The number is determined by equation (6):

$$s_i = \frac{t_{\max}}{t_i}\,, \tag{6}$$

which can be applied after a computational experiment, which takes place at the very beginning of the program, is completed. The experiment is organized in such a way that all the computational nodes compute (each one independently) a vector inner product. For each node the time of the computations is determined and called t_i. Of course, the slowest processor has the longest computation time t_{max}. So, as can be seen from equation (6), the coefficient of speed of particular processor is determined as a relation of the maximal time to the particular processor computation time.

4. Preconditioning

It has been shown in many publications (for instance, [3]) that by applying a preconditioning to system (1), it is possible to achieve better convergence of an iterative method. We present results of applying a simple preconditioning method, called "diagonal preconditioning", which is based on multiplying both sides of the system (1) by an inverted diagonal matrix that is constructed taking only the diagonal from matrix A (equation (7)). An advantage of this method is its simplicity, so that the additional time needed for the operation of preconditioning is very small. On the other hand, the method does not seem to improve considerably the convergence in many cases.

$$D^{-1}Ax = D^{-1}b\,. \qquad (7)$$

5. Speedup analysis

5.1 Testing matrices

Three different kinds of coefficient matrices have been used for testing presented PCGA.

The first type of a problem was to solve a 3D Laplace partial differential equation with Dirichlet boundary conditions. A distribution of an electrostatic potential inside a cube was searched. The potential had six different constant values on appropriate walls. The resulting coefficient matrix was a six-diagonal sparse matrix.

The second problem is described in [5] where the author computed eddy currents within ferromagnetic conductors exposed to transverse magnetic field, using a fixed point technique. The resulting matrices were also sparse.

The third problem was to compute a 2D magnetostatic field distribution in the middle of an electric machine tooth scale using FEM. In this case the resulting matrices were sparse too, but they were not so well conditioned as the previous ones.

5.2 PC cluster implementation

Both versions (with and without preconditioning) of the PCGA were tested in a PC cluster consisted of PC's having different computational power, connected by the Ethernet 100baseT network and on an SR-2201 Hitachi supercomputer. The results achieved in the cluster are presented below.

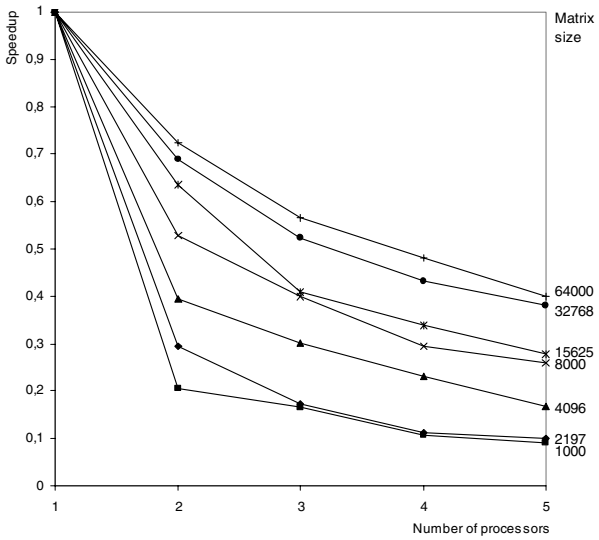

Figure 1. Computations speedup in the PC cluster implementation. Matrices from the first problem.

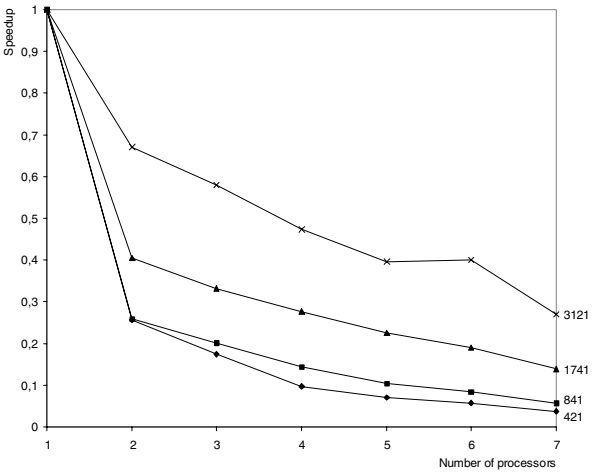

Figure 2. Computations speedup in the PC cluster implementation. Matrices from the second problem.

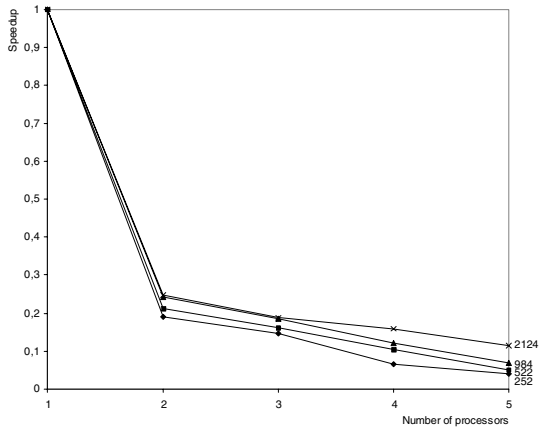

Figure 3. Computations speedup in the PC cluster implementation. Matrices from the third problem.

5.3 SR-2201 implementation

Here we present the results of speedup achieved on SR-2201 supercomputer.

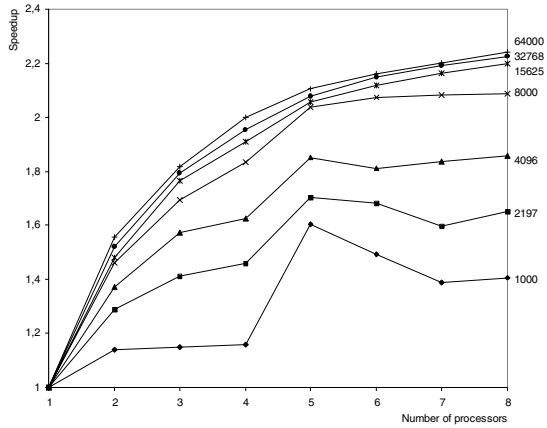

Figure 4. Computations speedup in the SR-2201 implementation. Matrices from the first problem.

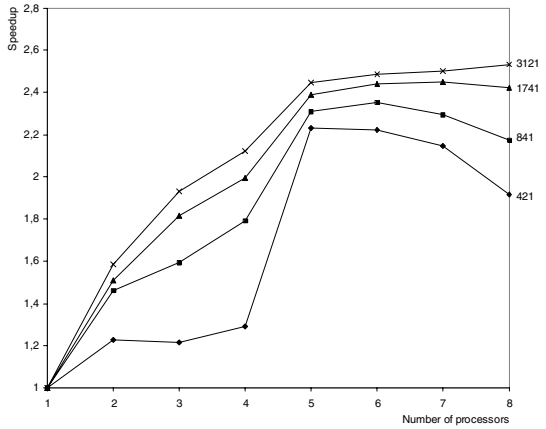

Figure 5. Computations speedup in the SR-2201 implementation. Matrices from the second problem.

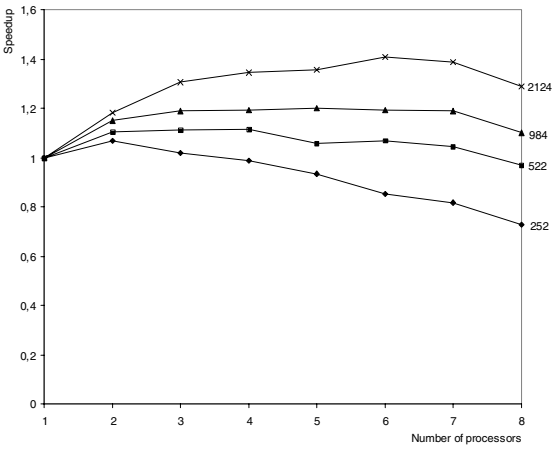

Figure 6. Computations speedup in the SR-2201 implementation. Matrices from the third problem.

6. Preconditioning

The PCGA has also been modified by applying a simple diagonal preconditioning in order to improve the convergence rate.

Table 1 shows the results of applying the preconditioner in all the three cases of problems mentioned in the previous chapter. In the table, S is a relation of the non-preconditioned version computation time to the preconditioned one – one could say it is actually a "speedup" of the computations provided by the preconditioning. Figure 7 presents a graphical representation of the data contained in the table.

Table 1. Computation time of the preconditioned and non-preconditioned PCGA on 1 processor

Matrix size	Non-preconditioned PCGA comp. time [s]	Preconditioned PCGA comp. time [s]	S
First problem case			
1000	0,11	0,11	1
2197	0,43	0,40	1,075
4096	1,84	1,43	1,287
8000	8,03	6,94	1,157
15625	25,54	22,46	1,137
32768	93,15	81,84	1,138
64000	286,17	253,75	1,128
Second problem case			
421	1,15	0,79	1,456
841	3,40	2,1	1,619
1741	20,52	12,48	1,644
3121	111,28	79,1	1,406
Third problem case			
252	43,09	1,00	43,09
522	157,39	2,01	78,30
984	909,19	4,03	225,61
2124	3400,00	14,67	231,77

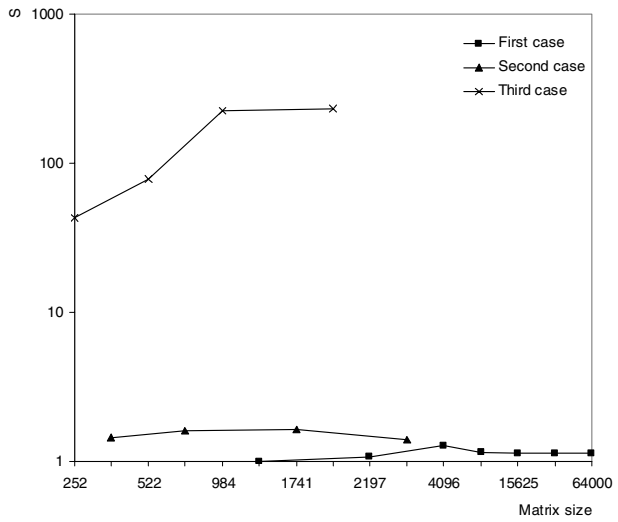

Figure 7. "Speedup" of the preconditioned algorithm.

7. Conclusions

The speedup of the parallel computations performed on the PC cluster is not satisfying. In all the three cases of the analyzed problems it is below 1. This is caused by the fact that the PCGA version presented in this paper contains many communication (data distribution and collection) operations in relation to the computations. The most time-consuming operations picked for parallelizing take considerably less time than in the PCGA version described in [1], because only the non-zero elements of the coefficient matrix are taken into account during the computations. Knowing that the amount of the non-zero elements does not extend the limit of about 1 percent or less of all the matrix elements, one can easily see there is not much to be computed during, for instance, matrix-vector multiplication.

The only advantage of the cluster implementation is therefore the possibility of performing computations of such large problems that would not fit on a single PC machine.

The speedup on the SR-2201 supercomputer is, however greater than 1. This is caused by the fact that the internal communication in the computer is much faster than in a 100-Base T Ethernet network.

When comparing the time of computations in the preconditioned and non-preconditioned version of the PCGA, one can see that the "speedup" (Fig. 7) provided by the preconditioning, strongly depends on the kind of computational problem. It has also been shown in [3], where a few preconditioners were investigated. After all, the "speedup" was always greater than 1 in the investigated cases, so one could draw a conclusion that it is always a good idea to apply preconditioning to the system (1). However, in particular cases, it is necessary to investigate if the applied preconditioning method does not cause the algorithm to lose its convergence at all.

This work has been done in the frame of the statutes work No. S/WE/4/98 and internal research grants of PJIIT.

7. References

[1] A. Jordan, R. P. Bycul, "The Parallel Algorithm of Conjugate Gradient Method", *Lecture Notes on Computer Science*, Vol. 2326, pp. 156-165, 2002.

[2] Jianming Jin, *The Finite Element Method in Electromagnetics*, John Wiley, New York, 1993.

[3] U. M. Yang, „Preconditioned Conjugate Gradient-Like Methods for Nonsymmetric Linear Systems", *CSRD-Report No. 1210*, University of Illinois at Urbana-Champaign, 1992.

[4] J. R. Shewchuk, "An Introduction to the Conjugate Gradient Method Without the Agonizing Pain", *School of Computer Science Carnegie Mellon University Pittsburgh*, August 1994.

[5] W. Peterson, "Fixed point technique in computing eddy currents within ferromagnetic conductors exposed to transverse magnetic field", *Archives of electrical engineering*, Vol. XLVII, no. 1, pp. 57-68, 1998.

[6] A. Jordan, "Theoretical Background of Parallel Electromagnetic Field Computations", *The 10th International Symposium on Applied Electromagnetics and Mechanics*, May 13-16, 2000, Tokyo, Japan, pp. 193-194.

[7] M. Cichomski, "Gradient parallel methods and evolutionary methods on supercomputer SR-2201", *bachelors thesis PJIIT*, Warsaw, Poland, 2001 (in polish).