INFORMATICA, 2005, Vol. 16, No. 3, 317–332

Analysis of Parallel Preconditioned Conjugate Gradient Algorithms

Raimondas ČIEGIS
Vilnius Gediminas Technical University
Saulėtekio al. 11, LT-10223 Vilnius, Lithuania
e-mail: rc@fm.vtu.lt

Received: December 2004

Abstract. The conjugate gradient method is an iterative technique used to solve systems of linear equations. The paper analyzes the performance of parallel preconditioned conjugate gradient algorithms. First, a theoretical model is proposed for estimation of the complexity of PPCG method and a scalability analysis is done for three different data decomposition cases. Computational experiments are done on IBM SP4 computer and some results are presented. It is shown that theoretical predictions agree well with computational results.

Key words: parallel algorithms, preconditioned conjugate gradient method, scalability analysis, incomplete factorization.

1. Introduction

Solution of many real-world and applied problems is reduced to the solution of large sparse systems of linear equations. Very often matrices of the systems have an additional regularity of non-zero coefficients, e.g., they are banded or block tridiagonal. Such kind of matrices is obtained after discretization of multidimensional self-adjoint elliptic partial differential equations via finite difference or finite volume methods.

Most frequently sparse systems of linear equations are solved by iterative methods such as the Conjugate Gradient (CG) algorithm. It is well-known that iterative methods can be parallelized much more simply than the direct methods. Therefore many papers are devoted to the development and analysis of parallel CG algorithms.

It follows from theory of iterative algorithms, that the most effective methods are preconditioned iterative algorithms. Let us assume that we solve a system of linear equations

$$AX = F\,, \tag{1}$$

where A is an $N \times N$ symmetric positive definite matrix, X and F are $N \times 1$ vectors. Then the preconditioned iterative method can be defined as a basic iterative method applied to the preconditioned system

$$B^{-1}AX = B^{-1}F\,.$$

This requires solving the system

$$BY = G\,, \tag{2}$$

where matrix B is called a preconditioner. A good parallel preconditioner needs to satisfy the following three requirements:

- Matrix B should be a good approximation of A in a sense that the condition number $\dfrac{\mu_u}{\mu_l}$, where μ_l, μ_u satisfy the estimates

  $$\mu_l B \leqslant A \leqslant \mu_u B\,,$$

  is much smaller than the condition number of the original matrix A.
- System (2) should be easy to solve, i.e., the number of arithmetical operations is of order $\mathcal{O}(N)$, where N is the number of equations in the system.
- The algorithm for solving (2) should have enough parallelism in order to be implemented efficiently on parallel computers with large number of processors.

In this paper we consider a system of linear equations arising from the three-dimensional Puasson problem

$$\begin{cases} -\sum\limits_{\alpha=1}^{3} \dfrac{\partial}{\partial x_\alpha}\Big(k(x)\dfrac{\partial u}{\partial x_\alpha}\Big) = f, & x \in Q := (0,1)\times(0,1)\times(0,1)\,, \\ u(x) = g, & x \in \partial Q\,. \end{cases} \tag{3}$$

via discretization by the finite-volume method on the uniform grid:

$$-a_{i+1,j,k}U_{i+1,j,k} - a_{i-1,j,k}U_{i-1,j,k} - a_{i,j+1,k}U_{i,j+1,k} - a_{i,j-1,k}U_{i,j-1,k}$$
$$-a_{i,j,k+1}U_{i,j,k+1} - a_{i,j,k-1}U_{i,j,k-1} + a_{ijk}U_{ijk} = F_{ijk}, \quad 1 \leqslant i,j,k \leqslant N\,.$$

The 7-point stencil is used for the approximation. The resulting system of equations may be written as

$$AU = F,$$

where A is a symmetric positive definite matrix. More details about the properties of such discrete approximations are given by Samarskii (2002), Golub and Van Loan (1996). For the following analysis it is sufficient to note that the obtained matrix A is block five-diagonal and there are seven non-zero elements in each row of the matrix.

First we develop a theoretical model, which estimates the complexity of the parallel PCG (PPCG) algorithm. Such performance tools are investigated for many parallel linear algebra algorithms, e.g., performance prediction tools for parallel Gauss algorithms are investigated by Čiegis *et al.* (2000). The main research efforts in this area are focused on heuristic methods for obtaining near-optimal solutions in a reasonable time (see Amoura

et al., 1998; Djordjević and Tošić, 1996). The scalability analysis of PPCG method was done in Gupta *et al.* (1997), where in particular a discretization of two-dimensional elliptic problem was investigated. Our goal is to generalize these results for three-dimensional problem and to test the accuracy of obtained theoretical estimates.

Second, we want to study the efficiency of some popular parallel preconditioners for solution of the obtained large sparse systems. We will consider the diagonal and the incomplete factorization IC preconditioning. The parallel version of IC is obtained by doing the factorization of a local part of the matrix at each processor. It is well known that such strategy reduces the convergence rate of the PCG algorithm, but a parallelism of the obtained preconditioner is the same as for the diagonal one. There have been many studies of the use of various ordering techniques to overcome the trade-off between parallelism and convergence in incomplete factorization. Some new multicolor orderings are proposed by D'Azevedo *et al.* (1992), Doi and Washio (1999), Monga–Made and Van der Vorst (2001). The comparison of parallel preconditioners for non-symmetric sparse linear systems is done by Ma (2004).

An exhaustive parallel library of sparse iterative methods and preconditioners in HPF and MPI was developed by Blanco *et al.* (2004). A model for predicting the performance of these codes is presented. The information offered by this model combines theoretical features of the methods and preconditioners in addition to certain practical considerations and predictions about aspects of the performance of their execution in distributed memory multiprocessors.

Software tools. A software package PCG is developed by Jourbert and Carey (1993) for solving systems of linear equations by means of preconditioned conjugate gradient-type iterative methods on a variety of computer architectures. The software is designed to give high performance with nearly identical user interface across different scalar, vector and parallel platforms as well as across different programming models such as shared memory, data parallel, and message passing programming interfaces.

Portable Library of Parallel Sparse Iterative Solvers (PSPARSLIB) is developed by Saad *et al.* (1998). It is a set of tools for solving large sparse linear systems on the distributed-memory computers. The library consists of the four major parts, the accelerators, preprocessing tools, preconditioning routines, and message-passing tools. The accelerators are based on the Krylov subspace methods, including CG method. The preconditioners provided with the library are tailored for preconditioning distributed sparse matrices, such as overlapping block Jacobi, multicolor block SOR, distributed ILU(0), approximate inverse preconditioners.

We also mention one more very interesting tool PETSc, which is developed by Balay *et al.* (2004). PETSc is a suite of data structures and routines for the scalable (parallel) solution of scientific applications modeled by partial differential equations. It employs the MPI standard for all message-passing communication. It includes many parallel solvers for systems of sparse linear equations.

Applications. Parallel PCG method is used to solve systems of linear equations in many applied projects. Here the main goal is to solve systems as large as possible in order to

obtain interesting simulation results. The efficiency of the parallel algorithm is not the prime interest in such studies. An example is given by Thiagarajan and Aravamuthan (2002). They describe a parallel implementation of an unstructured finite-element solver using the preconditioned PCG method. High-performance Fortran has been used for the implementation of the code. The PCG solver is set up for the element-by-element method. While this is a highly suitable method for the solution of very large problems, it is not inherently parallelizable in a distributed memory environment.

CG and parallel CG methods were used for numerical image smoothing by nonlinear diffusion filters (see Čiegis *et al.*, 2005).

We should mention that domain-decomposition (or data-decomposition) method is very popular in parallel computing. Many interesting industrial applications deal with solving optimization problems, e.g., global optimization, combinatorial optimization or more general discrete optimization problems. Parallel algorithms for solution of such problems are desribed by Pardalos *et al.* (1992), Pardalos *et al.* (1996). Very attractive are also randomized parallel algorithms, see, e.g., Pardalos and Rajasekaran (1999).

Our paper is organized as follows. In Section 2, we formulate the parallel preconditioned conjugate gradient method. The complexity analysis of PPCG method and scalability analysis of different data decompositions is presented in Section 3. In Section 4 the results of computational experiments with different preconditioners are presented. The test matrices were generated by the finite-volume approximation of the three dimensional Puasson equation. Some final conclusions are given in Section 5.

2. Formulation of the Parallel PCG Method

The serial PCG algorithm for solving a system of linear equations (1) can be formulated as follows (see Golub and Van Loan, 1996):

procedure `The serial PPCG algorithm`
begin

(1) $X^0, \quad n = 0, \quad R^0 = AX^0 - F,$
(2) $BW^0 = R^0, \quad P^0 = W^0\,.$
(3) **while** $\left((W^n, R^n) > \varepsilon\,(W^0, R^0) \right)$
(4) $G^n = AP^n\ ,$
(5) $\tau_{n+1} = \dfrac{(W^n, R^n)}{(G^n, P^n)}\,,$
(6) $X^{n+1} = X^n - \tau_{n+1} P^n\,,$
(7) $R^{n+1} = R^n - \tau_{n+1} G^n\,,$
(8) $BW^{n+1} = R^{n+1}\,.$
(9) $\beta_n = \dfrac{(W^{n+1}, R^{n+1})}{(W^n, R^n)}\,,$
(10) $P^{n+1} = W^{n+1} + \beta_n P^n\,,$
(11) $n := n + 1\,.$
(12) **end while**

end

Parallel PCG algorithm is obtained by using the data decomposition paradigm Kumar *et al.* (1994). We distribute vector X among processors, the vector F is aligned with X and distributed analogically. Then each processor is responsible for computations of the local part of vector X. Thus the parallel implementation of PCG method is identical to its serial implementation. As we will see the implementation of the parallel preconditioner can be different from the serial one.

Different operations of PPCG algorithm require different data communications between processors:

- Vector *saxpy* operations (see Steps (6), (7) and (10)) can be computed in parallel, when parameters τ_{n+1}, β_n are given. No communication between processors is needed, since all required data is locally available on each processor.
- Implementation of the matrix – vector multiplication requires additional information when boundary nodes of the local part of the vector X are updated (note, that these nodes are inner nodes in the global grid). Such information is obtained by exchanging data with neighbour processors in the specified topology of processors and the amount of data depends on the grid stencil, which is used to discretize the PDE model. We note that the communication step can be done in parallel. When all required information is saved in the local memory, the multiplication is performed locally on each processor.
- The computation of the inner product of two vectors require a global communication of all processors: first all processors compute inner products of local parts of vectors and then all local products are summed up. Different algorithms can be used to implement the global reduction step. We note that in MPI there exists a special function `MPI_ALLREDUCE`, which computes a sum and distributes it to all processors. It is assumed that MPI library is optimized for each type of supercomputer, taking into account specific details of the computer network.
- Parallelization of the algorithm for solving the system $BW = R$ depends on the structure of matrix B. If B is a diagonal preconditioner, then this step does not require any data communication and all computations are done locally and in parallel. If solving $B^{-1}R$ requires matrix-vector multiplications, then algorithms given above should be used.

3. Theoretical Model and Scalability Analysis

In this section we will develop a performance prediction model of the PPCG algorithm. It can help to answer the following questions:

- What type of data distribution between all processors is most effective for a given size of the problem N and number of processors p?
- What number of processors is optimal for a given system of linear equations and parallel computer?
- How scalable is the PPCG algorithm, i.e., how should the size of problem be increased with respect to the number of processors in order to maintain a certain efficiency?

3.1. *Complexity of the Serial PCG Algorithm*

As it follows from the given PCG algorithm, at each iteration we should compute two vector inner products at Steps (5) and (9), one matrix-vector multiplication at Step (4), and three vector *saxpy* operations (a combination of scalar-vector multiplication and vector addition) at Steps (6), (7), (10).

We will estimate the complexity of the PCG algorithm by counting the scalar *saxpy* operations (a combination of scalar-scalar multiplication and scalar addition).

Let assume that the size of the matrix A is $N \times N$ and there are m non-zero elements in each row of the sparse matrix. For the matrix, obtained by discretization of 3D elliptic problem, we have $N = n^3$, $m = 7$.

- The complexity of one vector *saxpy* operation is N.
- The vector inner product requires N operations.
- The matrix-vector multiplication can be performed in mN time.
- Let assume that the complexity of solving a system with preconditioning matrix B is bN (e.g., if B is a diagonal matrix, then $b = 1$).

As a result, the total complexity of the serial PCG algorithm can be expressed as

$$W = (5 + m + b)N\,. \tag{4}$$

3.2. *Complexity of the Parallel PCG Algorithm*

The nodes of the 3D grid can be partitioned among the processors by using different mappings. We will consider only simple Cartesian mappings, when the grid nodes are ordered naturally.

1D Mapping

The nodes of the x_1-dimension are distributed among processors by using a block distribution scheme. Then the size of largest local grid part, given to one processor, is equal to $\left\lceil \frac{n}{p} \right\rceil \times n \times n$, where $\lceil x \rceil$ denotes a smallest integer number not smaller than x.

The complexity of one parallel vector *saxpy* operation is given by

$$T_{1,p}(n) = \left\lceil \frac{n}{p} \right\rceil \times n \times n\,.$$

During matrix-vector multiplication each processor exchanges vector elements corresponding to its local boundary grid points in x_1-dimension with its two neighbouring processors. A total amount of data, exchanged between two processors, is equal to n^2 elements. This can be done in $\alpha + \beta n^2$ time, by using the *odd-even* data exchange algorithm. Here α is the message startup time and β is the time required to send one element of data.

Thus the complexity of the parallel matrix-vector multiplication is given by

$$T_{2,p}(n) = m\left\lceil \frac{n}{p} \right\rceil \times n \times n + \alpha + \beta n^2\,.$$

Parallel computation of the inner product of two vectors requires global communication among all processors during summation of local parts of the product. The complexity of such reduce operation depends strongly on the architecture of the parallel computer (see Hockney, 1991; Gupta *et al.*, 1997; Čiegis and Starikovičius, 2003).

We estimate the cost of broadcasting/reducing n items of data between p processors by

$$B(n,p) = R(p)(\alpha_b + \beta_b n),$$

where $R(p)$ depends on the algorithm used to implement the *ReduceAll* operation and the architecture of the computer. For the simplest algorithm we have $R(p) = p$. Thus the complexity of the parallel inner-product multiplication is given by

$$T_{3,p}(n) = \left\lceil \frac{n}{p} \right\rceil \times n \times n + R(p)(\alpha_b + \beta_b)\,.$$

Let assume that we use a diagonal preconditioner D, then the complexity of solving a system with $B = D$ is estimated by

$$T_{4,p}(n) = \left\lceil \frac{n}{p} \right\rceil \times n \times n\,.$$

Summing up the obtained estimates we compute the complexity of the parallel PCG algorithm

$$T_p(n) = (6+m)\left\lceil \frac{n}{p} \right\rceil \times n \times n + \alpha + \beta n^2 + R(p)(\alpha_b + \beta_b)\,. \tag{5}$$

This complexity estimate can be used as a performance prediction tool for the PPCG algorithm. Using the well-known Amdahl law (see, Lewis and El-Rewini 1992) we expect that the efficiency of a parallel algorithm should improve when a size of the problem is increased. It follows from (5) that the efficiency of the PPCG algorithm depends on the disbalance of sizes of the distributed local subgrids. Thus a size of the problem can grow, but the efficiency will degrade. Such situation will be illustrated in the next section, where results of computational experiments are presented.

Scalability Analysis

According to the definition of the isoefficiency function, we should find the rate at which the problem size W needs to grow with p for a fixed efficiency of the algorithm.

Let $H(p,W) = pT_p - W$ be the total overhead of a parallel algorithm. Then the isoefficiency function $W = g(p,E)$ is defined by the implicit equation (see Kumar *et al.*, 1994)

$$W = \frac{E}{1-E}\,H(p,W)\,. \tag{6}$$

For simplicity of notation we take $E = 0.5$. The scalability analysis is a very useful tool to predict the performance of the parallel algorithm (see, e.g., Čiegis et al., 2004).

The total overhead of the parallel PCG algorithm is given by

$$H(p,W) = (m+6)pn^2 + \alpha p + \beta pn^2 + R(p)p(\alpha_b + \beta_b)$$
$$= \alpha p + (m+6+\beta)p\left(\frac{W}{m+6}\right)^{\frac{2}{3}} + R(p)p(\alpha_b + \beta_b)\,.$$

The first term on the right-hand side of the equality arises due to the simple estimate

$$p\left\lceil\frac{n}{p}\right\rceil - n \leqslant p-1\,.$$

Since it is impossible to get the isoefficiency function in a closed form as a function of p, we will analyze the influence of each individual term. The component that requires the problem size to grow at the fastest rate determines the overall asymptotic isoefficiency function (see, Gupta *et al.*, 1997). After simple computations we get the following three isoefficiency functions

$$W = \mathcal{O}(p), \quad W = \mathcal{O}(p^3), \quad W = \mathcal{O}\big(pR(p)\big)\,.$$

Even for the most simple *ReduceAll* algorithm, when all processors send their local inner products to the master processor, which computes the sum and broadcasts the result to the other processors, the function $R(p) = p$. Thus the overall asymptotic isoefficiency function $W = \mathcal{O}(p^3)$ is determined by the second term (overheads due to matrix-vector multiplication and due to disbalances of sizes of local subgrids). It requires a cubic growth of problem size (or linear growth of n) with respect to p to maintain a certain efficiency.

2D Mapping

The nodes of the x_1, x_2-dimensions are distributed among processors using a block distribution scheme. Then the size of largest local grid part, given to one processor, is equal to $\left\lceil\frac{n}{\sqrt{p}}\right\rceil \times \left\lceil\frac{n}{\sqrt{p}}\right\rceil \times n\,.$

The complexity of one parallel vector *saxpy* operation is given by

$$T_{1,p}(n) = \left\lceil\frac{n}{\sqrt{p}}\right\rceil \times \left\lceil\frac{n}{\sqrt{p}}\right\rceil \times n\,.$$

During matrix-vector multiplication each processor exchanges vector elements corresponding to its local boundary grid points in x_1, x_2-dimensions with its four neighbouring processors. A total amount of data, exchanged between two processors, is equal to $\left\lceil\frac{n}{\sqrt{p}}\right\rceil \times n$ elements. Thus the complexity of the parallel matrix-vector multiplication is given by

$$T_{2,p}(n) = m\left\lceil\frac{n}{\sqrt{p}}\right\rceil \times \left\lceil\frac{n}{\sqrt{p}}\right\rceil \times n + 2\big(\alpha + \beta\left\lceil\frac{n}{\sqrt{p}}\right\rceil \times n\big)\,.$$

The complexity of a parallel inner-product multiplication is given by

$$T_{3,p}(n) = \left\lceil \frac{n}{\sqrt{p}} \right\rceil \times \left\lceil \frac{n}{\sqrt{p}} \right\rceil \times n + R(p)(\alpha_b + \beta_b)\,.$$

The complexity of solving a system with a diagonal preconditioner $B = D$ is estimated by

$$T_{4,p}(n) = \left\lceil \frac{n}{\sqrt{p}} \right\rceil \times \left\lceil \frac{n}{\sqrt{p}} \right\rceil \times n\,.$$

Summing up the obtained estimates we compute the complexity of the parallel PCG algorithm

$$T_p(n) = \big(m+6\big)\left\lceil \frac{n}{\sqrt{p}} \right\rceil \times \left\lceil \frac{n}{\sqrt{p}} \right\rceil \times n + 2\big(\alpha + \beta\left\lceil \frac{n}{\sqrt{p}} \right\rceil \times n\big) + R(p)(\alpha_b + \beta_b)\,.$$

Scalability Analysis
The total overhead of the parallel PCG algorithm is given by

$$\begin{aligned} H(p,W) &= 2\alpha p + (m+6+2\beta)pn + 2(m+6+\beta)\sqrt{p}n^2 + R(p)p(\alpha_b+\beta_b) \\ &= 2\alpha p + (m+6+2\beta)p\left(\frac{W}{m+6}\right)^{\frac{1}{3}} + 2\beta\sqrt{p}\left(\frac{W}{m+6}\right)^{\frac{2}{3}} \\ &\quad + R(p)p(\alpha_b+\beta_b)\,. \end{aligned}$$

We will analyze the influence of each individual term. After simple computations we get the following four isoefficiency functions

$$W = \mathcal{O}(p), \quad W = \mathcal{O}(p^{1.5}), \quad W = \mathcal{O}(p^{1.5}), \quad W = \mathcal{O}\big(pR(p)\big)\,.$$

If again we use a simple *ReduceAll* algorithm with $R(p) = p$, then the overall asymptotic isoefficiency function $W = \mathcal{O}(p^2)$ is determined by the overheads of the inner product computations.

Assuming that processors are connected by two-dimensional mesh $\sqrt{p} \times \sqrt{p}$, first all processors reduce local inner products along rows of the mesh, then the processors belonging to the first column of the mesh reduce their local sums along this column. The final result is broadcasted to all processors in inverse order. For such algorithm $R(p) = \sqrt{p}$ and the overall asymptotic isoefficiency function improves to $W = \mathcal{O}(p^{1.5})$. It requires growth of n with respect to p as $n = \mathcal{O}(\sqrt{p})$ to maintain a certain efficiency.

3D Mapping
The nodes of all three dimensions are distributed among processors using a block distribution scheme. Then the size of largest local grid part, given to one processor, is equal to $\left\lceil \frac{n}{\sqrt[1/3]{p}} \right\rceil \times \left\lceil \frac{n}{\sqrt[1/3]{p}} \right\rceil \times \left\lceil \frac{n}{\sqrt[1/3]{p}} \right\rceil$.

The complexity of one parallel vector *saxpy* operation is given by

$$T_{1,p}(n) = \left\lceil \frac{n}{\sqrt[3]{p}} \right\rceil \times \left\lceil \frac{n}{\sqrt[3]{p}} \right\rceil \times \left\lceil \frac{n}{\sqrt[3]{p}} \right\rceil .$$

During matrix-vector multiplication each processor exchanges vector elements corresponding to its local boundary grid points in three dimensions with its six neighbouring processors. A total amount of data, exchanged between two processors, is equal to $\left\lceil \frac{n}{\sqrt[3]{p}} \right\rceil \times \left\lceil \frac{n}{\sqrt[3]{p}} \right\rceil$ elements. Thus the complexity of the parallel matrix-vector multiplication is given by

$$T_{2,p}(n) = m\left\lceil \frac{n}{\sqrt[3]{p}} \right\rceil \times \left\lceil \frac{n}{\sqrt[3]{p}} \right\rceil \times \left\lceil \frac{n}{\sqrt[3]{p}} \right\rceil + 3\Big(\alpha + \beta\left\lceil \frac{n}{\sqrt[3]{p}} \right\rceil \times \left\lceil \frac{n}{\sqrt[3]{p}} \right\rceil\Big) .$$

The complexity of a parallel inner-product multiplication is given by

$$T_{3,p}(n) = \left\lceil \frac{n}{\sqrt[3]{p}} \right\rceil \times \left\lceil \frac{n}{\sqrt[3]{p}} \right\rceil \times \left\lceil \frac{n}{\sqrt[3]{p}} \right\rceil + R(p)(\alpha_b + \beta_b) .$$

The complexity of solving a system with a diagonal preconditioner $B = D$ is estimated by

$$T_{4,p}(n) = \left\lceil \frac{n}{\sqrt[3]{p}} \right\rceil \times \left\lceil \frac{n}{\sqrt[3]{p}} \right\rceil \times \left\lceil \frac{n}{\sqrt[3]{p}} \right\rceil .$$

Summing up the obtained estimates we compute the complexity of the parallel PCG algorithm

$$T_p(n) = \big(m+6\big)\left\lceil \frac{n}{\sqrt[3]{p}} \right\rceil^3 + 3\Big(\alpha + \beta\left\lceil \frac{n}{\sqrt[3]{p}} \right\rceil^2 \Big) + R(p)(\alpha_b + \beta_b) .$$

Scalability Analysis
The total overhead of the parallel PCG algorithm is given by

$$\begin{aligned} H(p,W) \;=\; & \big(m+6+3(\alpha+\beta)\big)p + 3(m+6+2\beta)p^{\frac{2}{3}} \left(\frac{W}{m+6}\right)^{\frac{1}{3}} \\ & + 3(m+6+\beta)p^{\frac{1}{3}} \left(\frac{W}{m+6}\right)^{\frac{2}{3}} + R(p)p(\alpha_b+\beta_b) . \end{aligned}$$

We will analyze the influence of each individual term. The first three terms are balanced if $W = \mathcal{O}(p)$, and the isoefficiency function due to overheads of the inner product is given by $W = \mathcal{O}\big(pR(p)\big)$. Thus the the overall asymptotic isoefficiency function is defined by the overheads of the parallel inner-product algorithm.

Let us assume that processors are connected by three-dimensional mesh $\sqrt[3]{p} \times \sqrt[3]{p} \times \sqrt[3]{p}$. Then the global *reduce* and *broadcast* operations can be implemented with

$R(p) = \sqrt[1/3]{p}$. Thus the problem size W has to grow as $\mathcal{O}\left(p^{\frac{4}{3}}\right)$ to maintain a certain efficiency.

The obtained scalability results show that the parallel PCG is a highly scalable algorithm. The efficiency of the algorithm is largest when we use three-dimensional mesh of processors. Then the asymptotic isoefficiency function is defined by the overheads of the parallel inner-product algorithm.

In applications the presented estimates of the complexity of PPCG algorithm can be used as a performance prediction tool to find the optimal topology of processors.

4. Computational Experiments with Different Parallel Preconditioners

In this section we present some results of computational experiments. Computations were performed on IBM SP4 computer at CINECA, Bologna. In all experiments we solved a system of linear equations which is obtained after the discretization of the three dimensional Puasson problem (3) by the finite-volume method.

Table 1 presents experimental speedup $S_p(n)$ and efficiency $E_p(n)$ values for solving problems of different size using the diagonal preconditioner $B = D$, where D is the main diagonal of the matrix A. 1D data decomposition was used in all computations.

As predicted by the scalability analysis, the efficiency of the PPCG algorithm degrades in the case of $p = 16$ processors for problems with $n = 39$ and $n = 119$, since the workload is divided not uniformly between processors. It also follows from the scalability analysis, that the efficiency should improve for $n = 127$, the results of computations confirm this theoretical prediction

$$S_{16}(127) = 13.4, \quad E_{16}(127) = 0.84\,.$$

Next we have tested how the efficiency of parallel PCG algorithm depends on multidimensional data mappings. The computations were performed on Intel Pentium 4 3.2 GHz PC cluster *Vilkas* at Vilnius Gediminas technical university. Our goal was to investigate how the efficiency of the parallel CG algorithm depends on the type of data decomposition for most simple communication networks. Table 2 presents CPU times

Table 1

Speedup and efficiency for the diagonal preconditioner: CPU time of the sequential PPCG algorithm (in *s*) $T_1(39) = 1.38$, $T_1(79) = 22.7$, $T_1(119) = 117.1$, and numbers of iterations $K(39) = 81$, $K(79) = 161$, $K(119) = 226$.

p	$S_p(39)$	$E_p(39)$	$S_p(79)$	$E_p(79)$	$S_p(119)$	$E_p(119)$
2	1.76	0.88	1.82	0.91	1.92	0.96
4	3.37	0.84	3.51	0.88	3.69	0.92
8	5.13	0.64	6.65	0.83	6.88	0.86
16	7.68	0.48	11.56	0.72	9.43	0.59

Table 2
CPU times for the diagonal preconditioner and $1D$, $2D$ and $3D$ domain decompositions

p	$T_p(120)$	$T_p(240)$	$T_p(360)$
$8 \times 1 \times 1$	10.08	98.99	481.9
$4 \times 2 \times 1$	9.26	91.12	466.3
$2 \times 2 \times 2$	9.48	93.34	472.8
$12 \times 1 \times 1$	7.70	68.97	324.3
$6 \times 2 \times 1$	6.92	63.13	315.3
$4 \times 3 \times 1$	6.86	62.80	315.7
$3 \times 2 \times 2$	6.87	63.93	322.9

$T_p(n)$ for solving problems of different size by using the diagonal preconditioner $B = D$ and different data decompositions. The sizes of the discrete problems are selected such that no load disbalance between local subproblems asigned to processors have arised.

It follows from the presented results, that the efficiency of the parallel CG algorithm is better for multidimensional data decompositions, but the dependence is not strong.

The IC Preconditioner

It is well known (see, Golub and Van Loan, 1996) that a diagonal preconditioner is not optimal. The number of iterations required to solve the system of linear equations can be reduced if a good preconditioner B is selected. In this section we consider one type of commonly used preconditioning matrix B, which is represented by the incomplete factorization $B = (D + L)D^{-1}(D + L^T)$, here L and D are the lower triangular and diagonal parts of the matrix A (for the other possible factorizations see Golub and Van Loan, 1996; Monga–Made and Vorst, 2001).

In practice the solution of system $BY = G$ is obtained first by solving a system $(D + L)V = G$ with a lower triangular matrix, and then by solving $(D + L^T)Y = DV$ with an upper triangular matrix. For the natural ordering of unknowns in Y both parts of the algorithm are fully sequential. A number of techniques were developed to attain parallelism in the solution of such systems. Mainly they are based on the reordering the system $AX = F$ and constructing an incomplete factorization for the reordered system (see, Monga–Made, 1995; Doi and Washio, 1999; Duff and Vorst, 1999; Monga–Made and Vorst, 2001).

First we consider a simple parallel version of IC preconditioner, when each processor computes the required factorization by using only its local part of the matrix A. Obviously, such algorithm can be implemented in parallel, but the convergence rate is decreased. We investigate experimentally the efficiency of such IC preconditioner. In Table 3, we give the performance of PPCG algorithm with the local IC preconditioner.

We see that the number of iterations for the PPCG algorithm with the local IC preconditioner grows up comparing it with the global IC preconditioner. As expected the spectral condition number of the matrix $B^{-1}A$ is bounded by $\mathcal{O}(n^2)$, while for the global IC preconditioner this number is of order $\mathcal{O}(n)$.

Table 3
Iteration numbers K_p and CPU times T_p for PPCG with the local IC preconditioner

p	$K_p(39)$	$T_p(39)$	$K_p(79)$	$T_p(79)$	$K_p(119)$	$T_p(119)$
1	19	0.72	29	8.78	36	45.81
2	38	0.68	73	11.2	101	56.26
4	31	0.30	60	4.77	86	23.9
8	28	0.15	54	2.31	78	10.6

Table 4
CPU times for PPCG with the diagonal preconditioner $T_{p,D}(n)$ and the local IC preconditioner $T_{p,IC}(n)$

p	$T_{p,D}(39)$	$T_{p,IC}(39)$	$T_{p,D}(79)$	$T_{p,IC}(79)$	$T_{p,D}(119)$	$T_{p,IC}(119)$
1	1.38	0.72	22.7	8.78	117.	45.8
2	0.79	0.68	12.6	11.2	61.2	56.3
4	0.41	0.30	6.51	4.77	31.7	23.9
8	0.27	0.15	3.42	2.31	17.1	10.6

The implementation of one iteration of the PPCG algorithm with IC preconditioner requires approximately twice more CPU time than for a diagonal preconditioner. But the parallel computation time of PPCG with the local IC preconditioner was smaller than the computation time using the diagonal preconditioner for all sizes of tested problems and numbers of processors. Some results are presented in Table 4.

Approximate IC Factorization

Let us consider one interesting modification of the IC factorization preconditioner. Kumar *et al.* (1994) proposed to increase the parallelism in IC factorization algorithm by using the truncated series

$$(I - L)^{-1}F \approx Y + LF + L^2F + \ldots + L^mF,$$

where I is the identity matrix and L is a strictly lower triangular matrix. Thus a solution of the system $(I - L)Y = F$ is changed to m matrix-vector multiplications. Such multiplications can be computed efficiently in parallel on most parallel systems.

A similar approach was used also by Akcadogan and Dag (2003), where the CG method was accelerated with an approximate inverse matrix preconditioner obtained from a linear combination of matrix-valued Chebyshev polynomials. We note that the largest eigenvalue of A should be known to implement this algorithm and the Power Method was employed to find an approximation of this eigenvalue.

Our goal is to test experimentally the quality of the obtained approximate IC preconditioner by solving the system of equations approximating 3D elliptic problem. In

Table 5

CPU times $T_1(n)$ and iteration numbers $K_1(n)$ for PPCG with the the approximate IC preconditioner, here m is a number of terms in truncated series

m	$K_1(79)$	$T_1(79)$	$K_1(119)$	$T_1(119)$
1	86	29.9	122	149.6
2	75	41.8	106	192.7
4	61	52.2	75	204.7
8	52	73.9	75	359.0
16	43	109.3	61	639.8
32	38	185.0	51	1044.

Table 6

CPU times T_p, speedups S_p and efficiency E_p for PPCG with the the approximate IC preconditioner, here $m = 4$

p	$T_p(119)$	$S_p(119)$	$E_p(119)$
2	119.6	1.71	0.86
4	63.67	3.22	0.81
8	32.27	6.34	0.79
16	19.17	10.68	0.67

Table 5, we present CPU times and numbers of iterations for different values of m. All computations were done for a serial version of the algorithm.

We see that for growing m the decrease in numbers of iterations do not compensate the increase in complexity of the one iteration, therefore the optimal value is $m = 1$. Nevertheless the parallelism of this preconditioner is quite good, as it follows from results presented in Table 6.

5. Conclusions

As it follows from computational results given above the measured speedups of the parallel PCG algorithm agree well with the predictions given by the theoretical model, which was developed to estimate the complexity of PPCG method. The scalability analysis shows that a three dimensional data decomposition should be used for computations with a large number of processors. A special attention should be given for the parallel implementation of the inner product of two vectors, since this part of the algorithm determines the overall asymptotic isoefficiency function.

Our results show that a parallel modification of the IC factorization preconditioner, which is constructed by using only the local part of the matrix A, is more efficient than

a simple diagonal precondioner. The implementation of such modification is based on a serial version of the IC factorization algorithm.

The IC factorization preconditioner which is based on truncated series is high scalable, but the overall efficiency of the obtained algorithm is worse than using the diagonal preconditioner.

Acknowledgments

The author thanks the referee for his constructive criticism which helped to improve the clarity of this note.

The work has been performed under the Project HPC-EUROPA (RII3-CT-2003-506079), with the support of the European Community – Research Infrastructure Action under the FP6 "Structuring the European Research Area" Programme. I gratefully acknowledge the hospitality and excellent working conditions in CINECA, Bologna. In particular I thank Dr. Giovanni Erbacci for his help.

This work also has been supported in part by Lithuanian State Science and Studies Foundation grant V-04049 and Eureka grant CTBSTROKE E!2981.

References

Akcadogan, C., H. Dag (2003). A parallel implementation of Chebyshev preconditioned conjugate gradient method. In *Proceedings of the Second International Symposium on Parallel and Distributed Computing*, IEEE, Computer Society. pp. 101–108.

Amoura, A., E. Bampis, J. König (1998). Scheduling algorithms for parallel Gaussian elimination with communication costs. *IEEE Transactions on Parallel and Distributed Systems*, **9**(7), 679–686.

D'Azevedo, E.F., P.A. Forsyth, W. Tang (1992). Ordering methods for preconditioned conjugate gradient methods applied to unstructured grid problems. *SIAM J. Matrix Anal. Appl.*, **10**(4), 567–584.

Balay, S., K. Buschelman, V. Eijkhout, W.D. Gropp, D. Kaushik, M.G. Knepley, L. Curfman McInnes, B.F. Smith, H. Zhang (2004). *PETSc Users Manual*. ANL-95/11 – Revision 2.1.5. Argonne National Laboratory.

Blanco, V., P. Gonzalez, J.C. Cabaleiro, D.B. Heras, T.F. Pena, J.J. Pombo, F.F. Rivera (2004). Performance prediction for parallel iterative solvers. *The Journal of Supercomputing*, **28**(2), 177–191.

Čiegis, R., A. Dement'ev, G. Šilko (2004). Parallel numerical modelling of short laser pulse compression. *Mathematical Modelling and Analysis*, **9**(2), 115–126.

Čiegis, R., A. Jakušev, A. Krylovas, O. Suboč (2005). Parallel algorithms for solution of nonlinear diffusion problems in image smoothing. *Mathematicall Modelling and Analysis*, **10**(2), 155–174.

Čiegis, R., V. Starikovičius (2003). Realistic performance tool for the parallel block LU factorization algorithm. *Informatica*, **14**(2), 167–180.

Čiegis, R., V.Starikovičius, J. Wasniewski (2000). On the efficiency of scheduling algorithms for parallel Gaussian elimination with communication delays. *Lecture Notes in Computer Science*, **1947**, PARA2000. In T. Surevik, F. Manne, R. Moe, A.H. Gebremedhin (Eds.), *The Fifth Workshop on Applied Parallel Computing*. Springer. pp. 75–82.

Djordjević, G., M. Tošić (1996). A heuristic for scheduling task graphs with communication delays onto multiprocessors. *Parallel Computing*, **22**, 1197–1214.

Doi, S., T. Washio (1999). Ordering strategies and related techniques to overcome the trade-off between parallelism and convergence in incomplete factorizations. *Parallel Computing*, **25**, 1995–2014.

Duff, I.S., H.A. van der Vorst (1999). Developments and trends in the parallel solution of linear systems. *Parallel Computing*, **25**, 1931–1970.

Golub, G.H., Ch. Van Loan (1996). *Matrix Computations*. The Johns Hopkins University Press, Baltimore.

Gupta, A., V. Kumar, A. Sameh (1997). Performance and scalability of preconditioned conjugate gradient methods on parallel computers. *IEEE Transactions on Parallel and Distributed Systems*, **6** (5), 455–469.

Jourbert, W.D., G.F. Carey (1993). PCG: A software package for the iterative solution of linear systems on scalar, vector and parallel computers. In R.F. Sincovec, D.E. Keyes, M.R. Leuze, L.R. Petzold, D.A. Reed (Eds.), *Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing*, *SIAM*. pp. 515–518.

Hockney, R. (1991). Performance parameters and benchmarking on supercomputers. *Parallel Computing*, **17**, 1111–1130.

Kumar, V., A. Grama, A. Gupta, G. Karypis (1994). *Introduction to Parallel Computing: Design and Analysis of Algorithms*. Benjamin/Cummings, Redwood City.

Lewis, T.G, H. El-Rewini (1992). *Introduction to Parallel Computing*. Prentice-Hall, Inc.

Ma, S. (2004). Comparisons of the parallel preconditioners for large nonsymmetric sparse linear systems on a parallel computer. *International Journal of High Speed Computing*, **12**(1), 55–68.

Monga–Made, M. (1995). Ordering strategies for modified block incomplete factorizations. *SIAM J. Sci. Stat. Comput.*, **16**, 378–399.

Monga–Made, M., H.A. van der Vorst (2001). A generalized domain decomposition paradigm for parallel incomplete LU factorization preconditionings. *Future Generation Computer Systems*, **17**, 925–932.

Pardalos, P.M., A. Philips, J.B. Rosen (1992). *Topics in Parallel Computing in Mathematical Programming*. Science Press.

Pardalos, P.M., Sanguthevar Rajasekaran (Eds.) (1999). *Advances in Randomized Parallel Computing*. Kluwer Academic Publishers.

Pardalos, P.M., G. Xue, P.D. Panagiotopoulos (1996). Parallel algorithms for global optimization. In A. Ferreira and P.M. Pardalos (Eds.), *Solving Combinatorial Optimization Problems in Parallel: Methods and Techniques*. Spinger-Verlag. *Lecture Notes in Computer Science*, **1054**, 233–247.

Saad, Y., G. Ching Lo, S. Kuznetsov (1998). *PSPARSLIB Users Manual: a Portable Library of Parallel Sparse Iterative Solvers*. `http://www-users.cs.umn.edu/ saad/software/p_sparslib/`

Samarskii, A.A. (2001). *The Theory of Difference Schemes*, Marcel Dekker, Inc., New York–Basel.

Thiagarajan, G., V. Aravamuthan (2002). Parallelization strategies for element-by-element preconditioned conjugate gradient solver using high-performance Fortran for unstructured finite-element applications on Linux clusters. *J. Comp. in Civ. Engrg.*, **16** (1), 1–10.

R. Čiegis has graduated from Vilnius University Faculty of Mathematics in 1982. He received the PhD degree from the Institute of Mathematics of Byelorussian Academy of Science in 1985 and the degree of Habil. Doctor of Mathematics from the Institute of Mathematics and Informatics, Vilnius in 1993. He is a professor and the head of Mathematical Modeling department of Vilnius Gediminas Technical University. His research interests include numerical methods for solving nonlinear PDE, parallel numerical methods and mathematical modeling in nonlinear optics, porous media flows, technology, image processing, biotechnology.

Lygiagrečiojo jungtinių gradientų algoritmo analizė

Raimondas Čiegis

Jungtinių gradientų metodas yra iteracinis algoritmas, kuris naudojamas tiesinių simetrinių lygčių sistemų sprendimui. Šiame darbe nagrinėjame lygiagretųjį neišreikštinį jungtinių gradientų algoritmą. Pirmiausia pateiktas teorinis modelis, leidžiantis įvertinti šio metodo skaičiavimo sudėtingumą ir atlikta algoritmo išplečiamumo analizė, kai duomenų matrica skaidoma naudojant vienmatę, dvimatę ir trimatę topologijas. Skaičiavimo eksperimentai atlikti IBM SP4 kompiuteriu, pateikiami šių eksperimentų rezultatai. Parodyta, kad jie artimi teorinio modelio prognozėms.