Project Report
HIPERGEOS COP–94 00820

Parallel Complexity of Conjugate Gradient Method with Circulant Preconditioners

Ivan Lirkov and Svetozar Margenov
BAS, Sofia, Bulgaria

Abstract

This report follows the research plan of the HIPERGEOS Project COP–94 00820.

The parallel numerical solution of large scale elliptic boundary value problems is discussed in this report. We analyze the parallel complexity of the circulant preconditioners when solving elliptic problems by Preconditioned Conjugate Gradient (PCG) method. Both methods, the conjugate gradient and the inversion of the circulant matrices using Fast Fourier Transform (FFT) are highly parallelizable.

The parallel algorithm is implemented on a cluster of workstations PVM. A set of parallel numerical test is presented at the end of the report. The obtained parallel speed–up and parallel efficiency can be evaluated as very encoraging.

Our next goal is to implement the studied here parallel circulant algorithms as a bisic preconditioning solver for 3D elasticity problems.

The experimental results presented in this report were obtained due to the active assistance of the project collaborative group of Prof. R. Beauwens from ULB Brussels.

1 Introduction

In this report we are concerned with the numerical solution of linear boundary value problems of elliptic type. After discretization, such problems are reduced to find the solution of linear systems of the form $Ax = b$. We shall only consider the case where A is symmetric and positive definite. In practice, large problems of this class are often solved by iterative methods, such as the conjugate gradient method. At each step of these iterative methods only the product of A with a given vector v is needed. Such methods are therefore ideally suited to exploit the sparsity of A.

Typically, the rate of convergence of these methods depends on the condition number $\kappa(A)$ of the coefficient matrix A: the smaller $\kappa(A)$ is, the faster convergence. Unfortunately, for elliptic problems of second order, usually $\kappa(A) = O(n^2)$, where n is the number of mesh points in each coordinate direction, and hence grows rapidly with n. To somewhat facilitate this problem, these methods are almost always used with a preconditioner C. The preconditioner is chosen with two criteria in mind: to minimize $\kappa(C^{-1}A)$ and to allow efficient computation of the product $C^{-1}v$ for a given

vector v. These two goals are often conflicting ones and much research has been done into devising preconditioners that strike a delicate balance between both.

One of the most popular and the most successful class of preconditioners is the class of incomplete LU (ILU) factorizations, see e.g. [9]. For some of these preconditioners, it can be proven that $\kappa(C^{-1}A) = O(n)$ for certain classes of elliptic problems, see [8, 10, 1]. This is a much slower growth compared to the unpreconditioned system.

One potential problem with the ILU preconditioners is that it have limited degree of parallelism. Some attempts to modify the method and to devise other more parallel methods often result in a deterioration of the convergence rate.

We use another class of preconditioners that is based on averaging the coefficients of A to form a circulant approximation. A part of our motivation is to exploit the fast inversion of circulant systems by FFT (see [7, 15]). In addition, the recent research on circulant preconditioners for Toeplitz systems [5, 4, 11, 12] shows good results for favorable clustering of eigenvalues of the preconditioned system.

Circulant preconditioners for Toeplitz systems have been proposed first by Olkin [16] and Strang [20]. Their properties and practicality have been studied by Strang [20], Chan & Jin [3], Chan & Olkin [6], etc.

In this report we consider three preconditioners that are constructed by circulant approximation of the initial matrix A. The first two are proposed by R.H.Chan and T.F.Chan in [2] and they are called block-circulant and point-circulant. They are based on a diagonal by diagonal averaging of the coefficients of the initial matrix. The block-circulant preconditioner preserves the block structure of the matrix A in contrast to the point-circulant one. The third one — the circulant block-factorization preconditioner is proposed by I. Lirkov, S. Margenov and P.S. Vassilevski in [13]. It is based on a circulant approximation of the blocks of A and in this way the sensitivity of the circulant approximation to varying coefficients is relaxed along one of the coordinate directions.

The relative condition number of the preconditioned system for a model Poisson problem for all of these preconditioners is estimated by $O(n)$, i.e., the same as for certain ILU type preconditioners. The inversion of these preconditioners is based on the fast inversion of the circulant matrices using FFT and they are highly parallelizable (see, e.g., [15, 17]).

The main goal of this study is analysis of the parallel complexity of the PCG method with considered circulant preconditioners. Because of the convergence rate of these preconditioners is asymptotically the same we estimate only the parallel execution time for one PCG iteration. We compare the parallel complexity, speed-up and parallel efficiency for the studied preconditioners on various standard architectures. Finally, we demonstrate some numerical results obtained on a cluster of RISK6000.

2 The elliptic problem and the circulant preconditioners

We consider the following 2D elliptic problem:

$$-(a(x,y)u_x)_x - (b(x,y)u_y)_y = f(x,y) \tag{1}$$

on the unit square $[0,1]\times[0,1]$ with Dirichlet boundary condition. Let the domain be discretized by an uniform grid with n grid points in each coordinate direction, denoted by x_i and y_j. Consider the usual five-point centered difference approximation. This discretization leads to a system of linear algebraic equations

$$Ax = b. \tag{2}$$

If the grid points are ordered along, e.g., the y-direction first, the matrix A admits a block-tridiagonal structure where the diagonal blocks are tridiagonal matrices and the off-diagonal blocks are diagonal matrices. The matrix A can be written in the following form

$$A = tridiag(-A_{i,i-1}, A_{i,i}, -A_{i,i+1}) \qquad i = 1,2,\ldots,n,$$

where

$$\begin{aligned} A_{i,i} &= tridiag(-a_{j,j-1}, a_{j,j}, -a_{j,j+1}), \quad j = (i-1)n+1,\ldots,in, \quad i = 1,2,\ldots,n,\\ A_{i,i+1} &= diag(a_{j,j+n}), \quad j = (i-1)n+1,\ldots,in, \quad i = 1,\ldots,n-1,\\ A_{i,i-1} &= diag(a_{j,j-n}), \quad j = (i-1)n+1,\ldots,in, \quad i = 2,\ldots,n. \end{aligned}$$

Let us remind that a circulant matrix C has the form $(C_{k,j}) = \left(c_{(j-k) \bmod m}\right)$, where m is the size of C. Let us also denote for any given coefficients $(c_0, c_1, \ldots, c_{m-1})$ by $C = (c_0, c_1, \ldots, c_{m-1})$ the circulant matrix

$$\begin{bmatrix} c_0 & c_1 & c_2 & \ldots & c_{m-1} \\ c_{m-1} & c_0 & c_1 & \ldots & c_{m-2} \\ \vdots & \vdots & \vdots & & \vdots \\ c_1 & c_2 & \ldots & c_{m-1} & c_0 \end{bmatrix}.$$

Any circulant matrix can be factorized as

$$C = F\Lambda F^*, \tag{3}$$

where Λ is a diagonal matrix containing the eigenvalues of C, and F is the Fourier matrix

$$F = \frac{1}{\sqrt{m}} \left\{ e^{2\pi \frac{jk}{m}\mathbf{i}} \right\}_{0\le j,k\le m-1}.$$

Here $\mathbf{i}$ stands for the imaginary unit.

Let us denote

$$\bar{a} = \frac{1}{n^2} \sum_{i=1}^{n-1} \sum_{j=(i-1)n+1}^{in} a_{j,j+n}$$
$$\bar{b} = \frac{1}{n^2} \sum_{i=1}^{n} \sum_{j=(i-1)n+1}^{in-1} a_{j,j+1}$$

$$\begin{aligned} C_0 &= (2\bar{a} + 2\bar{b} + 2\rho n^{-2}, -\bar{b}, 0, \ldots 0, -\bar{b}) \\ C_1 &= -\bar{a}I = (-\bar{a}, 0, \ldots 0) \end{aligned}$$

where ρ is a positive constant independent of n. Then the *block-circulant preconditioner* can be written in the following form

$$C_B = (C_0, C_1, 0, \ldots 0, C_1) \tag{4}$$

and the *point-circulant preconditioner* can be written as

$$C_P = (c_0^P, c_1^P, \ldots c_{n^2-1}^P) \tag{5}$$

where

$$c_0^P = 2(\bar{a} + \bar{b}) + \rho n^{-2}, \quad c_1^P = c_{n^2-1}^P = -\bar{b}, \quad c_n^P = c_{n^2-n}^P = -\bar{a}.$$

The general form of the *circulant block-factorization preconditioner* is defined by

$$C_F = tridiag(-C_{i,i-1}, C_{i,i}, -C_{i,i+1}) \qquad i = 1, 2, \ldots, n, \tag{6}$$

where $C_{i,j} = Circulant(A_{i,j})$ is a circulant approximation of the corresponding block $A_{i,j}$.

3 Model of the computation and communication times

We assume that the computations and communications are not overlapped and therefore, the execution time of the parallel implementation is the sum of the computation time and the communication time.

We shall use in our analysis standard models for the arithmetic and communication times [18]. First, assuming no arithmetic vectorization, the execution of M arithmetic operations on one processor takes time

$$T_a = M * t_a, \tag{7}$$

where t_a is the average unit time to perform one arithmetic operation on one processor. Second, the local communication time of transfer of M data from one processor to its neighbor is approximated by

$$T_{local} = t_s + M * t_c, \tag{8}$$

where t_s is the start-up time and t_c is the incremental time necessary for each of all M words to be sent.

Finally, for the various architectures, we denote the following two communication times, respectively by:

- Broadcasting a number from one processor to all others: $b(p)$, where there are p processors in the computer system;
- Gathering p data packets, each packet with M/p words, in one processor from all others: $g(M,p)$.

In this report we discuss three architecture models: ring, square grid and hypercube. For these architectures the above communication times are respectively:

	$b(p)$	$g(M,p)$
ring	$(t_s + t_c)\frac{p}{2}$	$p\,t_s + M\,t_c$
grid	$(t_s + t_c)\sqrt{p}$	$2\sqrt{p}\,t_s + M(1 + \frac{1}{\sqrt{p}})\,t_c$
hypercube	$(t_s + t_c)\log p$	$\log p\,t_s + M(1 - \frac{1}{p})\,t_c$

To achieve a low cost of the processor synchronization and communication, we partition the unit square into p rectangles so that each rectangle contains n/p y-grid lines. We map all grid points in one rectangle onto a processor so that the values at the grid points in neighbor rectangle are computed on the neighbor processor. In this way, the CG method requires a communication of only n values from one grid line with neighbor processors.

4 Model analysis of the parallel execution times

We analyze the total execution time of a parallel system of p processors to solve problem (2) by a Preconditioned Conjugate Gradient method using three circulant preconditioners. We shall estimate the execution time of performing one PCG iteration. The algorithm includes a multiplication of the matrix A by a vector, an inversion of the preconditioner (solving a system of equations with matrix C), two inner products and three linked triads (scalar times a vector and an addition of vectors).

$$T_{PCG} = T_{mult} + T_{prec} + 2T_{inn_prod} + 3T_{linked_triads}$$

$$\begin{aligned} T_{mult} &= 9\frac{n^2}{p}t_a + 4(t_s + nt_c) \\ T_{inn_prod} &= 2\frac{n^2}{p}t_a + g(p,p) + b(p) \\ T_{linked_triads} &= 2\frac{n^2}{p}t_a \end{aligned}$$

and T_{prec} for the discussed preconditioners is

$$T_{prec} = \begin{cases} T_{BC} = 40\frac{n^2}{p}\log n t_a + 2\frac{n^2}{p}t_a + 2g(\frac{n^2}{p},p) & \text{for block-circulant} \\ T_{PC} = 40\frac{n^2}{p}\log n t_a + 14\frac{n^2}{p}t_a + 2g(\frac{n^2}{p},p) & \text{for point-circulant} \\ T_{CBF} = 20\frac{n^2}{p}\log n t_a + 12\frac{n^2}{p}t_a + 2g(\frac{n^2}{p},p) & \text{for circulant block-factorization} \end{cases}$$

Finally, we obtain the following execution times for the studied architectures and preconditioners

	block-circulant	point-circulant	circ. block-fact.
ring	$2T_1 + 21T_2 + T_3 + T^r$	$2T_1 + 33T_2 + T_3 + T^r$	$T_1 + 31T_2 + T_3 + T^r$
grid	$2T_1 + 21T_2 + T_3 + T^g$	$2T_1 + 33T_2 + T_3 + T^g$	$T_1 + 31T_2 + T_3 + T^g$
hypercube	$2T_1 + 21T_2 + T_3 + T^h$	$2T_1 + 33T_2 + T_3 + T^h$	$T_1 + 31T_2 + T_3 + T^h$

where

$$
\begin{aligned}
T_1 &= 20\frac{n^2}{p}\log n t_a \\
T_2 &= \frac{n^2}{p}t_a \\
T_3 &= 2\frac{n^2}{p}t_c + 4nt_c + 4t_s \\
T^r &= p(5t_s + 3t_c) \\
T^g &= 2\sqrt{p}(5t_s + 2t_c) + 2(\frac{n^2}{p^{\frac{3}{2}}} + p)t_c \\
T^h &= 2\log p(3t_s + t_c) + 2(p - \frac{n^2}{p^2} - 1)t_c
\end{aligned}
$$

We determine the communication start-up time $t_s = 196.94 * 10^{-6}s$ and the elemental communication time $t_c = 0.3574 * 10^{-6}s$ from the numerical results on Intel iPSC/860 provided by Smith [19]. In the following figures, the flop rate per unit time is defined as 10 MFLOPS, i.e., $t_a = 10^{-7}s$.

In Figures 1–3 we graph the total execution time of one PCG iteration as a function of the problem size for 64 processors. For large problems the arithmetic time increases faster than the communication one and the dependence of the time on the architecture decreases. In Figures 4–6 we show the execution time as a function of the number of processors for $n = 4096$. As it was predicted, for relatively large problems the execution times on various architectures differ a little. All times are shown in seconds. Figures 7–9 show the parallel efficiency ($E_p = \frac{T_1}{pT_p}$) of one PCG iteration for $n = 64$ and we again denote that for small problem size the execution time depends strongly on the architecture.

The execution time T_{PCG} for CBF preconditioner is smaller than for other preconditioners for each choice of the grid size and the number of processors. Vice versa, there are very small differences between the execution times for the block-circulant and the point-circulant preconditioners. We pay attention on that for some problems (such as periodic, anisotropic, etc., see [13, 14]) CBF preconditioner possesses some advantages and respectively the number of iterations is smaller than for other circulant preconditioners. We conclude that the total parallel execution time is the smallest for the CBF preconditioner.

All figures demonstrate that the hypercube is better choice for PCG algorithm. However, from a practical point of view the ring architecture is very simple for realization and very often it can be more convenient for a parallel execution.

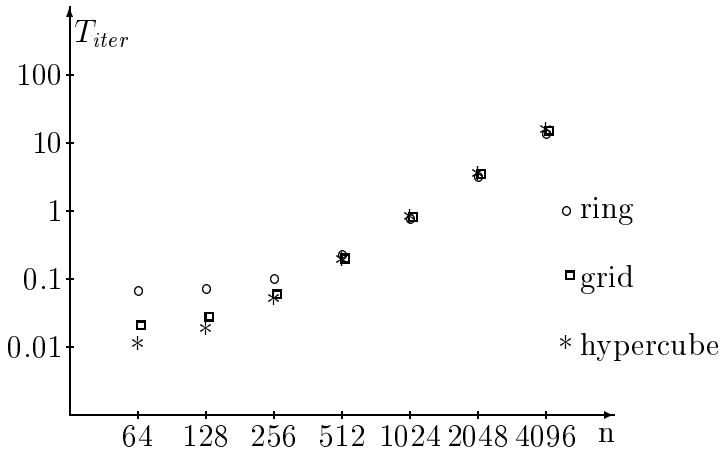

Figure 1: The execution time of performing one PCG iteration with block-circulant preconditioner for $p = 64$.

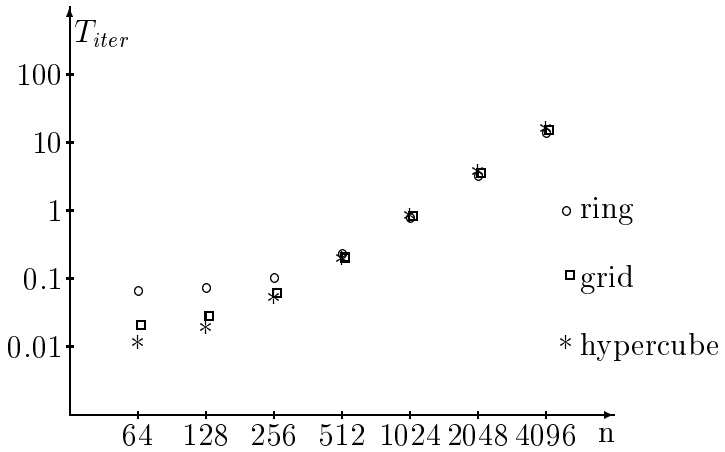

Figure 2: The execution time of performing one PCG iteration with point-circulant preconditioner for $p = 64$.

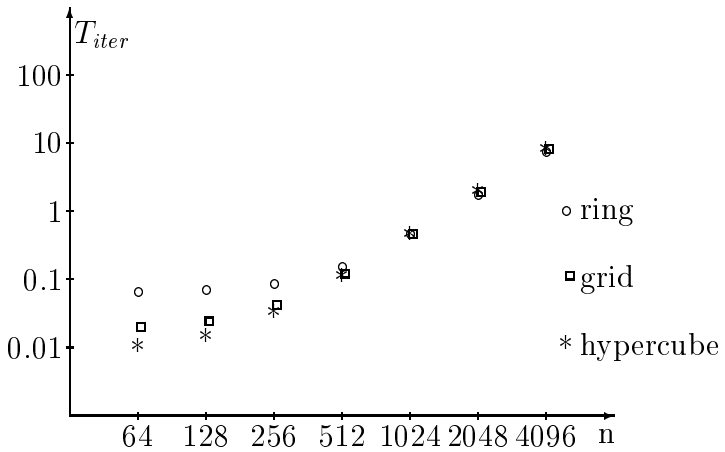

Figure 3: The execution time of performing one PCG iteration with circulant block-factorization preconditioner for $p = 64$.

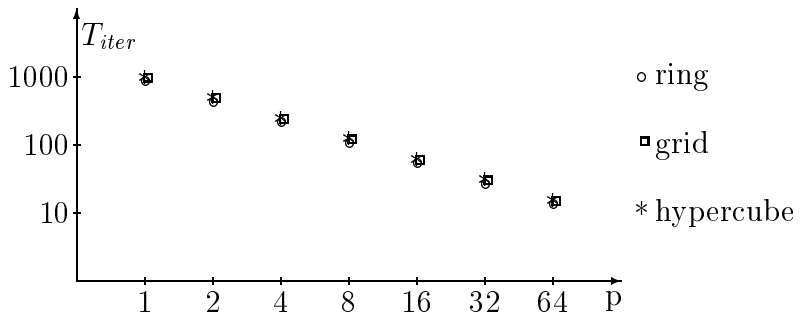

Figure 4: The execution time of performing one PCG iteration with block-circulant preconditioner for $n = 4096$.

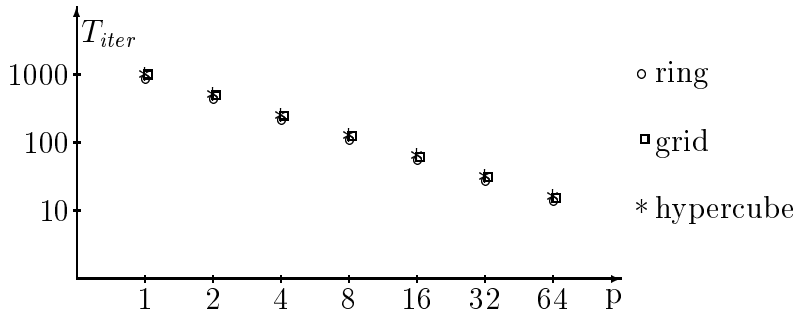

Figure 5: The execution time of performing one PCG iteration with point-circulant preconditioner for $n = 4096$.

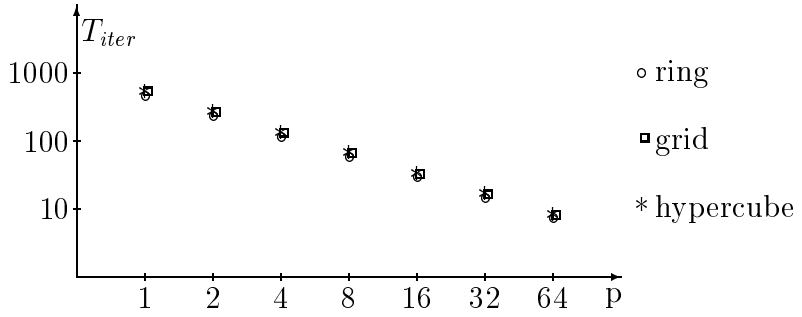

Figure 6: The execution time of performing one PCG iteration with circulant block-factorization preconditioner for $n = 4096$.

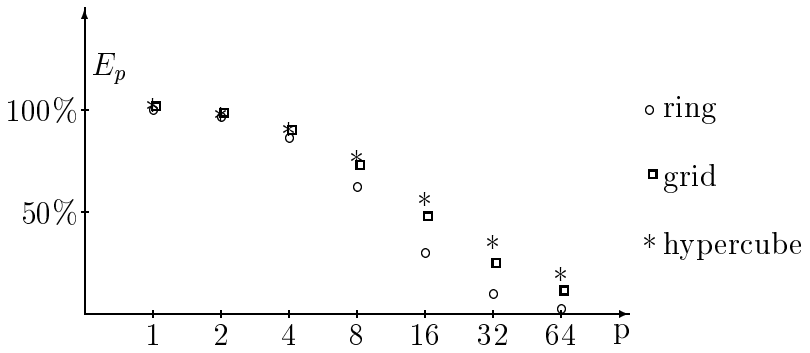

Figure 7: The parallel efficiency of performing one PCG iteration with block-circulant preconditioner for $n = 64$.

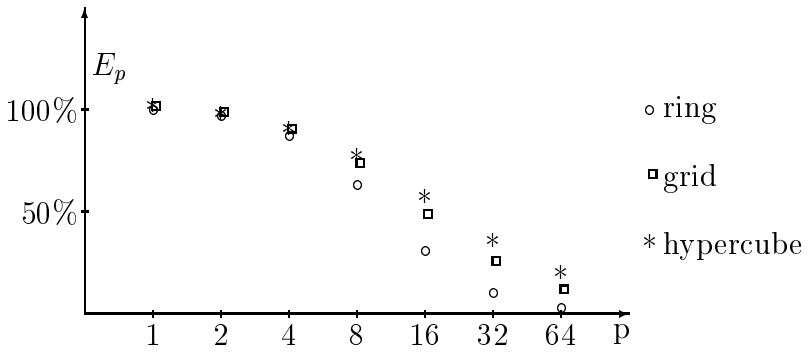

Figure 8: The parallel efficiency of performing one PCG iteration with point-circulant preconditioner for $n = 64$.

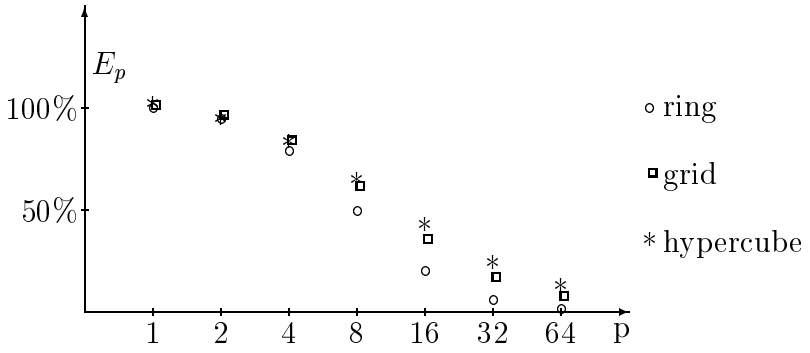

Figure 9: The parallel efficiency of performing one PCG iteration with circulant block-factorization preconditioner for $n = 64$.

5 Numerical tests

We test the parallel implementation of the PCG algorithm on a cluster of eight machines RISK6000. In Table 1 we show the real execution time for one PCG iteration, the speed-up ($S_p = \frac{T_1}{T_p}$) and the parallel efficiency ($E_p = \frac{S_p}{p}$).

Table 1: Iteration time, speed-up and paralleling efficiency.

n	p	T_p	S_p	E_p
8	1	0.0018182		
	2	0.0050000	0.36	18.18%
	4	0.0027273	0.67	16.67%
	8	0.0130000	0.14	1.75%
16	1	0.0042857		
	2	0.0050000	0.86	42.86%
	4	0.0053846	0.80	19.90%
	8	0.0153846	0.28	3.48%
32	1	0.0166667		
	2	0.0158824	1.05	52.47%
	4	0.0152632	1.09	27.30%
	8	0.0226316	0.74	9.21%
64	1	0.0702020		
	2	0.0536364	1.31	65.44%
	4	0.0400000	1.76	43.88%
	8	0.0315000	2.23	27.86%
128	1	0.3125930		
	2	0.2240740	1.40	69.75%
	4	0.1450000	2.16	53.90%
	8	0.0948148	3.30	41.21%
256	1	1.3219200		
	2	0.9073530	1.46	72.84%
	4	0.5432350	2.43	60.84%
	8	0.3084780	4.29	53.57%
512	1	5.9108800		
	2	4.0834800	1.45	72.38%
	4	2.4468900	2.42	60.39%
	8	1.3094000	4.51	56.43%

We point your attention that, from practical point of view, the presented results are very reasonable for large n, and in particular for $p \leq 8$. The increase of the influence of the communication time with p, on the speed-up, and on the efficiency, is also well expressed.

6 Summary

We are concerned with the numerical solution of elliptic linear boundary value problems. After discretization, such problems reduce to the solution of linear systems. We use a preconditioner based on a circulant approximation of the initial matrix. We exploit the fast inversion of circulant systems.

The computation and the inversion of these circulant preconditioners are highly parallelizable on a wide variety of architectures. For the used timing model we conclude in general that for relatively large problems or for relatively small number of processors the influence of the architecture on the execution time is negligible. The speed-up and the parallel efficiency increase with the size of the discrete problem.

References

[1] O. Axelsson and V.A. Barker, *Finite Element Solution of Boundary Value Problems: Theory and Computation* (Academic Press, Orlando, Fl., 1983).

[2] R.H. Chan and T.F. Chan, Circulant preconditioners for elliptic problems, *J. Numerical Lin.Alg.Appl.* **1** (Mar. 1992) 77-101.

[3] R. Chan and X. Jin, A family of block preconditioners for block systems, *SIAM J.Sci.Stat.Comp.* **13**, 5 (Sep. 1992) 1218-1235.

[4] R. Chan and G. Strang, Toeplitz equations by conjugate gradients with circulant preconditioner, *SIAM J.Sci.Stat.Comp.* **10** (1989) 104-119.

[5] T.F. Chan, An Optimal Circulant Preconditioner for Toeplitz Systems, *SIAM J.Sci. Stat.Comp.* **9** (Jul. 1988) 766-771.

[6] T.F. Chan and J.A. Olkin, Preconditioners for Toeplitz-block matrices, *Presented at SIAM Conference on Linear Algebra in San Francisco* (1990).

[7] P.J. Davis, *Circulant matrices*, John Wiley, New York (1979).

[8] T. Dupont, R. Kendall and H. Rachford, An Approximate Factorization Procedure for Solving Self-Adjoint Elliptic Difference Equations, *SIAM J.Numer.Anal.* **23**, 1 (Feb. 1986) 44-57.

[9] G.H. Golub and C.F. Van Loan, *Matrix computations*, 2nd edition (John Hopkins Univ.Press, Baltimore, 1989).

[10] I. Gustafsson, A Class of First Order Factorization Methods, *BIT* **18** (1978) 142-156.

[11] T. Huckle, Circulant and skewcirculant matrices for solving Toeplitz matrix problems, *SIAM J.Matr.Anal.Appl.*, **13** (1992), 767–777.

[12] T. Huckle, Some aspects of circulant preconditioners, *SIAM J.Sci. Comput.*, **14** (1993), 531–541.

[13] I. Lirkov, S. Margenov, P.S. Vassilevski, Circulant block-factorization preconditioners for elliptic problems, *Computing*, **53** 1 (1994), 59–74.

[14] I. Lirkov, S. Margenov, L. Zikatanov, Circulant Block-Factorization Preconditioning of Anisotropic Elliptic Problems, UCLA CAM Report **95–39**, 1995.

[15] Charles Van Loan, *Computational frameworks for Fast Fourier Transform*, (SIAM, Philadelphia, 1992).

[16] J.A. Olkin, Linear and nonlinear deconvolution problems, Ph. D. Thesis, Rice University, TX, 1986.

[17] M. Pester, S. Rjasanow, A Parallel Version of the Preconditioned Conjugate Gradient Method for Boundary Element Equations, *Numer. Lin. Alg. with Appl.*, **2** 1 (1995), 1–16.

[18] Y. Saad, M.H. Schultz, Data Communication in Parallel Architectures, *Parallel Comput.*, 11 (1989), 131–150.

[19] B.F. Smith, A parallel implementation of an iterative substructuring algorithm for problems in three dimensions, *SIAM J.Sci.Stat.Comp.* **14** (1993), 406–423.

[20] G. Strang, A proposal for Toeplitz matrix calculations, *Stud. Appl.Math.* **74** (1986) 171-176.