Chapter 1
Parallel Conjugate Gradient Method with Circulant Block-Factorization Preconditioners for 3D Elliptic Problems

I. Lirkov* S. Margenov* M. Paprzycki† J. McLaughlin†

Abstract

Parallel performance of a new solver for 3D elliptic problems based on a circulant block-factorization preconditioner is investigated. Experimental data collected on a number of parallel computers is reported and discussed.

1 Introduction

Let us consider the numerical solution of a self-adjoint second order 3D linear boundary value problem of elliptic type. After discretization, such a problem results in a linear system $Ax = b$, where A is a structured sparse symmetric positive definite matrix. In the computational practice, large-scale problems of this class are most often solved by the Krylov subspace based iterative methods (e.g. conjugate gradient method). Each step of such a method requires only a product of A with a given vector v allowing one to exploit the sparsity of A. The rate of convergence of these methods depends on the condition number κ of the coefficient matrix A: smaller $\kappa(A)$ leads to faster convergence. Unfortunately, for the second order 3D elliptic problems, typically $\kappa(A) = \mathcal{O}(N^{2/3})$, where N is the size of the discrete problem, and hence grows rapidly with N. To alleviate this problem, iterative methods are almost always used with a preconditioner M. The preconditioner is chosen with two criteria in mind: to minimize $\kappa(M^{-1}A)$ and to allow efficient computation of the product $M^{-1}v$ for any given vector v. These two goals are often in conflict and a large body of research exists devoted to devising preconditioners that strike a balance between the two. Recently, a third aspect has been added to considerations, namely, the efficiency of applying the iterative method (and thus the preconditioner) on a parallel computer.

We focus our considerations on a model 3D case where the computational domain is regular (a cube). It is important to note that the development of new high performance methods for large- and very large-scale problems of this class is strongly motivated by their applications as a part of the general framework of domain decomposition and patched local refinement algorithms in very general settings including time dependent and/or nonlinear problems.

One of the most popular and the most successful preconditioners used by the state-of-the-art iterative solvers are the incomplete LU (ILU) factorizations. One potential problem with the ILU preconditioners is that they have a rather limited degree of parallelism. Attempts to modify the approach and introduce more parallelism often result in a deterioration of the convergence rate. In 1992, R. Chan and T. F. Chan [3] proposed

*CLPP, BAS, Acad. G. Bonchev 25A, 1113 Sofia, Bulgaria ({ivan,margenov}@bas.bg)
†Dept. of CSS, USM, Hattiesburg, MS 39406-5106 ({marcin,mclaughe}@orca.st.usm.edu)

another class of preconditioners which is based on averaging coefficients of A to form a block-circulant approximation (see also [4, 5]). The block-circulant preconditioners are highly parallelizable but they are very sensitive to a possible high variation of the coefficients of the elliptic operator. To reduce this sensitivity a new class of circulant block-factorization (CBF) preconditioners was introduced in 1994 by Lirkov, Margenov and Vassilevski [6].

The main goal of this note is to report on the parallel performance of the PCG method with a new circulant block-factorization preconditioner applied to a model 3D linear PDE of elliptic type (the preconditioner was initially introduced in [8, 9]). The results of experiments performed on the SGI Cray Origin2000, HP-Convex Exemplar SPP-2000 (X-Class), Cray J-9x, Cray T3E and Sun Ultra-Enterprise high performance computers are presented and analyzed.

We proceed as follows. In Section 2 we sketch the algorithm of the parallel preconditioner (for more details see [8, 9]). Section 3 contains the theoretical estimate of its arithmetical complexity. Finally, in Section 4 we report the results of our experiments.

2 Circulant block factorization

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 dimension of C. Let us also denote by $C = (c_0, c_1, \ldots, c_{m-1})$ the circulant matrix with the first row $(c_0, c_1, \ldots, c_{m-1})$. Any circulant matrix can be factorized as $C = F\Lambda F^*$ where Λ is a diagonal matrix containing the eigenvalues of C, and F is the Fourier matrix of the form

$$F_{jk} = \frac{1}{\sqrt{m}} e^{2\pi \frac{jk}{m} i}, \tag{1}$$

where $F^* = \overline{F}^T$ denotes the adjoint matrix of F.

The CBF preconditioning technique incorporates the circulant approximations into the framework of LU block factorization. Let us consider a 3D elliptic problem (see also [8]):

$$-(a(x_1, x_2, x_3)u_{x_1})_{x_1} - (b(x_1, x_2, x_3)u_{x_2})_{x_2} - (c(x_1, x_2, x_3)u_{x_3})_{x_3} = f(x_1, x_2, x_3)$$

on the unit cube $[0, 1] \times [0, 1] \times [0, 1]$ with Dirichlet boundary conditions. If the domain is discretized on a uniform grid with n_1, n_2 and n_3 grid points along the coordinate directions, and if a standard (for such a problem) seven-point FDM (FEM) approximation is used, then the stiffness matrix A admits a block-tridiagonal structure. The matrix A can be thus written in the form

$$A = tridiag(-A_{i,i-1}, A_{i,i}, -A_{i,i+1}) \qquad i = 1, 2, \ldots, n_1,$$

where $A_{i,i}$ are block-tridiagonal matrices which correspond to the x_1-plane and the off-diagonal blocks are diagonal matrices.

In this case the general CBF preconditioning approach is applied to construct the preconditioner M_{CBF} in the form

$$M_{CBF} = tridiag(-C_{i,i-1}, C_{i,i}, -C_{i,i+1}) \qquad i = 1, 2, \ldots n_1, \tag{2}$$

where $C_{i,j} = \text{Block-Circulant}(A_{i,j})$ is a block-circulant approximation of the corresponding block $A_{i,j}$. The relative condition number of the CBF preconditioner for the model (Laplace) 3D problem for $n_1 = n_2 = n_3 = n$ is (for derivation see [8]):

$$\kappa(M_0^{-1} A_0) \leq 4n. \tag{3}$$

2.1 Parallel circulant block-factorization preconditioner

The basic advantage of circulant preconditioners is their inherent parallelism. Let us now describe how to implement in parallel an application of the inverse of the preconditioner to a given vector. Using the standard LU factorization procedure, we can first split $M = D - L - U$ into its block-diagonal and strictly block-triangular parts respectively. Then the *exact* block-factorization of M can be written in the form

$$M = (X - L)(I - X^{-1}U),$$

where $X = diag(X_1, X_2, \ldots, X_n)$ and the blocks X_i are determined by the recursion

$$X_1 = C_{1,1}, \quad \text{and} \quad X_i = C_{i,i} - C_{i,i-1}X_{i-1}^{-1}C_{i-1,i}, \qquad i = 2, \ldots, n_1. \tag{4}$$

It is easy to observe here that X_i are also block-circulant matrices.

In order to compute $M^{-1}v$ we rewrite the block-circulant blocks of the preconditioner as

$$C_{i,j} = (F \otimes F)\Lambda_{i,j}(F^* \otimes F^*).$$

Here $\otimes$ denotes the Kronecker product. It can be observed that for X_i in (4) we have

$$X_i = (F \otimes F)D_i^{-1}(F^* \otimes F^*)$$

and the latter yields

$$\begin{aligned} D_1^{-1} &= \Lambda_{1,1}, \\ D_i^{-1} &= \Lambda_{i,i} - \Lambda_{i,i-1}D_{i-1}\Lambda_{i-1,i}. \end{aligned}$$

Let $\Lambda = tridiag(\Lambda_{i,i-1}, \Lambda_{i,i}, \Lambda_{i,i+1})$. Then the following relation holds

$$Mu = v \qquad \Longleftrightarrow \qquad (I \otimes F \otimes F)\Lambda(I \otimes F^* \otimes F^*)u = v.$$

The above system can be rewritten as

$$\begin{pmatrix} \mathcal{F} & & & & \\ & \mathcal{F} & & & \\ & & \mathcal{F} & & \\ & & & \ddots & \\ & & & & \mathcal{F} \end{pmatrix} \begin{pmatrix} \Lambda_{11} & \Lambda_{12} & & & \\ \Lambda_{21} & \Lambda_{22} & \Lambda_{23} & & \\ & \Lambda_{32} & \Lambda_{33} & & \\ & & & \ddots & \\ & & & & \Lambda_{nn} \end{pmatrix} \begin{pmatrix} \mathcal{F}^* & & & & \\ & \mathcal{F}^* & & & \\ & & \mathcal{F}^* & & \\ & & & \ddots & \\ & & & & \mathcal{F}^* \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_n \end{pmatrix} = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_n \end{pmatrix}$$

where $\mathcal{F} = F \otimes F$.

We can distinguish three stages in computing $u = M^{-1}v$:

$$\begin{aligned} &(1) \qquad \hat{v} = (I \otimes F^* \otimes F^*)v \\ &(2) \qquad \Lambda\hat{u} = \hat{v} \\ &(3) \qquad u = (I \otimes F \otimes F)\hat{u}. \end{aligned} \tag{5}$$

Due to the special form of F (see (1) above), we can use a fast Fourier transform to perform the first and third stages of the algorithm. Namely, we use a standard two-dimensional block-FFT which is easily parallelizable (see [12]). The second stage consist of solving two recurrence equations

$$\left| \begin{array}{l} \hat{w}_1 = D_1\hat{v}_1 \\ \hat{w}_i = D_i(\hat{v}_i - \Lambda_{i,i-1}\hat{w}_{i-1}) \\ \qquad i = 2, 3, \ldots n_1 \end{array} \right. \qquad \left| \begin{array}{l} \hat{u}_n = \hat{w}_n \\ \hat{u}_i = \hat{w}_i - D_i\Lambda_{i,i+1}\hat{u}_{i+1} \\ i = n_1 - 1, n_1 - 2, \ldots 1 \end{array} \right. \tag{6}$$

Since blocks D_i and $\Lambda_{i,j}$ in the recurrences (6) are diagonal the solution of n independent linear systems can be calculated in parallel.

3 Parallel complexity

Let us present the theoretical estimate of the total execution time T_{PCG} for one PCG iteration for the proposed circulant block-factorization preconditioner on a parallel system with p processors (detailed analysis of parallel complexity can be found in [9]). Each iteration consists of one matrix vector multiplication with matrix A, one multiplication with the inverse of the preconditioner M_{CBF} (solving a system of equations with matrix M), two inner products and three linked triads (a vector updated by a vector multiplied by a scalar). Consequently

$$T_{PCG}(p) = T_{mult} + T_{prec} + 2T_{inn\_prod} + 3T_{triads}.$$

For simplicity we assume that the mesh sizes are equal and they are equal to an exact power of two, i.e., $n_1 = n_2 = n_3 = n = 2^l$. We also assume that the time to execute K arithmetic operations on one processor is $T_a = K * t_a$, where t_a is an average time of one arithmetic operation. In addition, the communication time of a transfer of K words between two neighbor processors is $T_{local} = t_s + K * t_c$, where t_s is the start-up time and t_c is the time for each word to be sent/received. Finally, let us assume that a 2-radix algorithm is used to calculate the FFT's and thus the cost per processor is $T_{FFT}(n) = 5n \log n t_a$. Then the formula for computational complexity has the form

$$T_{PCG}(p) \quad = \quad 5\,(7 + 4\log n)\,\frac{n^3}{p}t_a + 4\left(t_s + n^2 t_c\right) + 2g(\frac{n^3}{p}, p) + 2g(p,p) + 2b(p),$$

where $b(p)$ denotes time to broadcast a number from one processor to all other processors and $g(K,p)$ denotes time to gather $\frac{K}{p}$ words from all processors into one processor. It can be shown that, when the appropriate frmulas for g and b are derived and substituted into the expression above and when only the leading terms in n are taken into consideration then e.g. for the shared memory parallel computer we obtain:

$$T_{PCG}(p) \approx 2pt_s + 2(1 - \frac{1}{p})\frac{n^3}{p}t_c + 5(7 + 4\log n)\frac{n^3}{p}t_a, \tag{7}$$

and the formula for the speedup can be derived

$$S_p \approx \frac{5(7 + 4\log n)}{2\frac{p^2}{n^3}\frac{t_s}{t_a} + 2(1 - \frac{1}{p})\frac{t_c}{t_a} + 5(7 + 4\log n)}p. \tag{8}$$

Obviously, $\lim_{n\to\infty} S_p = p$ and $\lim_{n\to\infty} E_p = 1$, i.e., the algorithm is asymptotically optimal. More precisely, if $\log n \gg \frac{p^2}{n^3}\frac{t_s}{t_a} + \frac{t_c}{t_a}$, then E_p is near to 1. Unfortunately, the start-up time t_s is usually much larger than t_a, and for relatively small n the first term of the denominator in (8) is significant, and the efficiency is much smaller than 1.

4 Experimental results

In this section we report the results of the experiments executed on the SGI Cray Origin2000, HP-Convex Exemplar SPP-2000 (X-Class), Cray J-9x, Cray T3E and the Sun Ultra-Enterprise high performance computers. The results obtained on the latter system have been originally reported in [9]. We will report them here in the final section for the comparison with other systems. The code has been implemented in C and the parallelization has been facilitated by the MPI library [17, 16]. In all cases the manufacturer provided

TABLE 1
Parallel performance of the SGI Origin

	$n = 40$		$n = 48$	
p	*time*	*speedup*	*time*	*speedup*
1	0.46		0.79	
2	0.24	1.91	0.43	1.84
3			0.28	2.78
4	0.12	3.73	0.21	3.75
5	0.10	4.60		
6			0.14	5.4
8	0.07	6.86	0.11	7.03

TABLE 2
Parallel performance of the HP-Convex SPP-2000

	$n = 40$		$n = 48$	
p	*time*	*speedup*	*time*	*speedup*
1	0.57		1.01	
2	0.28	2.03	0.49	2.06
3			0.32	3.15
4	0.14	4.07	0.24	4.07
5	0.11	4.87		
6			0.16	5.97
8	0.08	7.31	0.14	7.42

MPI kernels have been used. No machine-dependent optimization has been applied to the code itself e.g. exactly the same code has been used on all machines. Instead, in all cases, the most aggressive optimization options of the compiler have been turned on. Times have been collected using the MPI provided timer. For all experiments we report the best results from multiple runs in interactive and batch modes on machines with varying workloads.

4.1 SGI Cray Origin2000

The results have been gathered on a system at NCSA in Urbana [13]. It is a dynamic shared memory computer with MIPS R10000 processors running at 250MHz (theoretical peak performance of 360 Mflops per processor). Table 1 presents results for $n = 40, 48$ and for $p = 1, \ldots, 8$ processors. Since our implementation is in experimental stages the problem size must be divisible by the number of processors and thus some numbers of processors were not used. The execution time (in seconds) and the relative speedup are reported.

The results are rather encouraging. For this, relatively small, problem the efficiency on 8 processors reaches 87% and is increasing as with the problem size.

4.2 HP-Convex Exemplar SPP-2000 (X-Class)

The results have been collected on the system at NCSA in Urbana. It is a dynamic shared memory computer with PA-RISC 8000 processors running at 180 MHz (with a theoretical peak performance of 720 Mflops). As above, Table 2 depicts the execution times and the relative speedup for $n = 40, 48$ and $p = 1, \ldots, 8$ processors.

The results viewed in isolation seem very positive as the 8-processor efficiency for the larger problem reaches 92%. However, when they are compared with these obtained on the SGI machine (which is based on a similar hardvare model), they are rather disappointing. A more detailed performance comparison between machines is presented in Section 4.4

Table 3
Parallel performance of the Cray J-9x

	$n = 40$		$n = 48$	
p	*time*	*speedup*	*time*	*speedup*
1	4.26		6.68	
2	2.22	1.91	3.47	1.92
3			2.36	2.83
4	1.16	3.67	1.83	3.65
5	0.96	4.43		
6			1.40	4.77
8	0.67	6.31	1.04	6.42

Table 4
Parallel performance of the Cray T3E

	$n = 40$		$n = 48$	
p	*time*	*speedup*	*time*	*speedup*
1	0.74		1.25	
2	0.36	2.02	0.62	2.01
3			0.41	3.04
4	0.18	4.06	0.31	4.16
5	0.15	5.02		
6			0.21	6.03
8	0.09	7.86	0.16	8.01

below. The superliner speedup should be attributed to the memory management. When the number of processors increases, the size of data per-processor decreases and thus reduces the burden on the cache memory and the cache management software. This is particularly important on the Exemplar architecture ([1, 11]).

4.3 Cray J-9x

The experiments have been run on the Cray at NPACI in Austin [14]. It is a shared memory vector-computer with processors running at theoretical peak of 200 MFlops. As previously, Table 3, depicts times and the relative speedup for the same problem sizes and numbers of processors.

Again, if it was not for the remaining results presented here, the parallelization obtained on the Cray seems reasonable (efficiency of 80% on 8 processors). It is, however, clear that the way that the code is implemented is not well suited for the vector-processing model of computation. The slightly smaller speedup can be partially attributed to the bus-based architecture (see also [10, 11]), and to the fact that when the experiments were run the workload on the machine was rather high.

4.4 Cray T3E

The final series of experiments has been run on a Cray T3E at NPACI in Austin. It is a distributed memory machine with DEC Alpha 21164 processors running at 400 MHz (theoretical peak performance of 500 Mflops). Table 4 summarizes the performance for the same problem sizes and numbers of processors as for the remaining machines.

The performance is quite similar to that of the SGI and Convex machines which is rather disappointing considering the per-processor peak performance. Interestingly, as on the Exemplar, we again observe superlinear speedup. This can be related to the very

TABLE 5
Performance comparison between the machines

	Origin		Exemplar		J-9x		T3E		Enterprise	
p	time	sp-up	time	sp-up	time	sp-up	time	sp-up	time	sp-up
1	1.34		1.68		13.34		2.19		6.03	
2	0.73	1.84	0.83	2.02	6.88	1.93	1.09	2.00	2.99	2.01
4	0.36	3.72	0.41	4.09	3.74	3.56	0.53	4.13	1.50	4.02
7	0.21	6.26	0.25	6.72	2.48	5.37	0.31	7.06	0.86	6.97
8	0.18	7.08	0.22	7.63	2.26	5.90	0.27	8.11	0.77	7.86

small amount of memory available on the machine in Austin (only 128 Mbytes per node) hampering the single processor performance.

4.5 Comparison between machines

Finally, in this section we compare the performance of all computers for the largest problem we have experimented with so far; $n = 56$. Table 5 contains the time and relative speedup reported for all five computers. It should be recalled that the Sun Ultra-Enterprise server is a symmetric multiprocessing system with UltraSPARC-II processors running at 336 MHz (theoretical peak performance of 500 Mflops).

The results follow trends observed earlier. The current implementation of the algorithm is clearly not well suited for the vector-processors of the Cray J-9x. The performance of the Sun machine cannot compete with the remaining three computers (which is very surprising considering the per-processor peak performance of the UltraSPARC-II nodes). The performance of the Cray T3E is hampered by the insufficient memory per node, as evidenced by the increasing degree of the superlinear speedup, and thus the results reported here do not represent the full capability of the architecture. The Exemplar is underperforming due to its old cache technology. While the MIPS processors use a two-way cache, the PA-RISC chips use a one-way cache [1]. This explains why a machine with twice the theoretical peak performance is slower. Overall, for the relatively small problem size and for the small number of processors used, the code parallelizes well.

5 Summary

We have presented initial assessment of the parallel performance of a new preconditioner for the linear systems arising from discretizations of linear elliptic PDEs in 3D. We were able to establish that the code parallelizes well and holds promise for the solution of larger systems on larger numbers of processors. In the near future we plan to explore this direction further.

Acknowledgments

This research has been supported by Bulgarian NSF Grant I-701/97. Computer time grants from NCSA and NPACI are kindly acknowledged.

References

[1] G. ASTFALK, Convex Computers, Personal communication.
[2] A. O. H. AXELSSON AND M. G. NEYTCHEVA, *Supercomputers and Numerical Linear Algebra*, KUN, Nijmegen, 1997.
[3] R. H. CHAN AND T. F. CHAN, *Circulant preconditioners for elliptic problems*, J. Num. Lin. Alg. Appl., 1 (1992), pp. 77–101.

[4] S. Holmgren and K. Otto, *Iterative solution methods for block-tridiagonal systems of equations*, SIAM J. Matr. Anal. Appl., 13 (1992), pp. 863–886.
[5] T. Huckle, *Some aspects of circulant preconditioners*, SIAM J. Sci. Comput., 14 (1993), pp. 531–541.
[6] I. Lirkov, S. Margenov, and P. S. Vassilevski, *Circulant block-factorization preconditioners for elliptic problems*, Computing, 53(1) (1994), pp. 59–74.
[7] I. Lirkov and S. Margenov, *Parallel complexity of conjugate gradient method with circulant preconditioners*, in Proceedings of the Parcella'96, R. Volmar, W. Erhard, and V. Jossifov, eds, Mathematical research, 96, Akademie Verlag, Berlin, 1996, pp. 279–286.
[8] I. Lirkov and S. Margenov, *Conjugate gradient method with circulant block-factorization preconditioners for 3D elliptic problems*, in Proceedings of the Workshop # 3 Copernicus 94-0820 "HIPERGEOS" project meeting, Rožnov pod Radhoštěm, 1996.
[9] I. Lirkov and S. Margenov, *Parallel complexity of conjugate gradient method with circulant block-factorization preconditioners for 3D elliptic problems*, in Recent Advances in Numerical Methods and Applications, O. P. Iliev, M. S. Kaschiev, Bl. Sendov, P. V. Vassilevski, eds., World scientific, Singapore, pp. 455–463, to appear.
[10] I. Lirkov, S. Margenov, and M. Paprzycki, *Parallel Solution of 2D Elliptic PDE's on Silicon Graphics Supercomputers*, in Proceedings of the 10th International Conference on Parallel and Distributed Computing and Systems, Y. Pan, et.al. (eds.), IASTED/ACTA Press, Anaheim, 1998, pp. 575-580.
[11] I. Lirkov, S. Margenov, and M. Paprzycki, *Benchmarking performance of parallel computers using a 2D elliptic solver*, in Recent Advances in Numerical Methods and Applications, O. P. Iliev, M. S. Kaschiev, Bl. Sendov, P. V. Vassilevski, eds., World scientific, Singapore, pp. 464–472, to appear.
[12] C. Van Loan, *Computational Frameworks for the Fast Fourier Transform*, SIAM, Philadelphia, 1992.
[13] National Center for Supercomputer Applications, http://www.ncsa.edu.
[14] National Partnership for Advanced Computational Infrastructure, http://www.npaci.edu.
[15] Y. Saad and M. H. Schultz, *Data communication in parallel architectures*, Parallel Comput., 11 (1989), pp. 131–150.
[16] M. Snir, St. Otto, St. Huss-Lederman, D. Walker, and J. Dongara, *MPI: The Complete Reference*, Scientific and engineering computation series. The MIT Press, Cambridge, Massachusetts, 1997, Second printing.
[17] D. Walker and J. Dongara, *MPI: a standard Message Passing Interface*, Supercomputer, 63 (1996), pp. 56–68.