Parallel Performance of a 3D Elliptic Solver

Ivan Lirkov[1], Svetozar Margenov[1], and Marcin Paprzycki[2]

[1] Central Laboratory for Parallel Processing, Bulgarian Academy of Sciences
Acad.G.Bonchev, block 25A, 1113 Sofia, Bulgaria
{ivan,margenov}@cantor.bas.bg
[2] Department of Computer Science and Statistics, University of Southern Mississippi,
Hattiesburg, Mississippi, 39406-5106, USA
marcin@orca.st.usm.edu

Abstract. It was recently shown that block-circulant preconditioners applied to a conjugate gradient method used to solve structured sparse linear systems arising from 2D or 3D elliptic problems have good numerical properties and a potential for high parallel efficiency. In this note parallel performance of a circulant block-factorization based preconditioner applied to a 3D model problem is investigated. The aim of the presentation is to report on the experimental data obtained on SUN Enterprise 3000, SGI/Cray Origin 2000, Cray J-9x, Cray T3E computers and on two PC clusters.

1 Introduction

Let us consider 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 $A\mathbf{x} = \mathbf{b}$, where A is a sparse symmetric positive definite matrix. In the computational practice, large-scale problems of this class are most often solved by Krylov subspace iterative (e.g. conjugate gradient) methods. Each step of such a method requires only a single matrix-vector product and allows exploitation of sparsity of A. The rate of convergence of these methods depends on the condition number κ of the matrix A (smaller $\kappa(A)$ results in faster convergence). Unfortunately, for second order 3D elliptic problems, usually $\kappa(A) = \mathcal{O}(N^{2/3})$, where N is the size of the discrete problem, and hence it 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 $M^{-1}\mathbf{v}$ for any given vector $\mathbf{v}$. These two goals are often in conflict and a lot of research has been done devising preconditioners that strike a balance between them. Recently, a third aspect has been added to the above two, namely, the parallel efficiency of the iterative method (and thus the preconditioner).

One of the most popular and the most successful preconditioners are the incomplete LU (ILU) factorizations. Unfortunately, standard ILU preconditioners have limited degree of parallelism. Some attempts to modify them and introduce more parallelism often result in a deterioration of the convergence rate. R. Chan

L. Vulkov, J. Waśniewski, and P. Yalamov (Eds.): NAA 2000, LNCS 1988, pp. 535–543, 2001.

and T. F. Chan [2] proposed another class of preconditioners based on averaging coefficients of A to form a block-circulant approximation. 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 [5] was introduced by Lirkov, Margenov and Vassilevski. Recently a new CBF preconditioner for 3D problems was introduced in [3,4].

The main goal of this note is to report on the parallel performance of the PCG method with a circulant block-factorization preconditioner applied to a model 3D linear PDE of elliptic type. Results of experiments performed on Sun Ultra-Enterprise, Crays J-9x and T3E, SGI/Cray Origin 2000 high performance computers and on two PC clusters are presented and analyzed.

We proceed as follows. In Section 2 we sketch the algorithm of the parallel preconditioner (for more details see [3,4]). 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 recall 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

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

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 [3]) on the unit cube 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 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 $C_{i,j}$ = Block-Circulant$(A_{i,j})$ is a block-circulant approximation of the corresponding block $A_{i,j}$. The stiffness matrix A and the preconditioner M_{CBF}

are $N \times N$ matrices where $N = n_1 n_2 n_3$. 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 [3]):

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

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 $M^{-1}\mathbf{v}$ 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 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

$$M\mathbf{u} = \mathbf{v} \qquad \Longleftrightarrow \qquad (I \otimes F \otimes F)\Lambda(I \otimes F^* \otimes F^*)\mathbf{u} = \mathbf{v}.$$

The above system can be rewritten as

$$\begin{pmatrix} \mathcal{F} & & & & \\ & \mathcal{F} & & & \\ & & \mathcal{F} & & \\ & & & \ddots & \\ & & & & \mathcal{F} \end{pmatrix} \begin{pmatrix} \Lambda_{11} & \Lambda_{12} & & & \\ \Lambda_{21} & \Lambda_{22} & \Lambda_{23} & & \\ & \Lambda_{32} & \Lambda_{33} & & \\ & & & \ddots & \\ & & & & \Lambda_{nn} \end{pmatrix} \begin{pmatrix} \mathcal{F}^* & & & & \\ & \mathcal{F}^* & & & \\ & & \mathcal{F}^* & & \\ & & & \ddots & \\ & & & & \mathcal{F}^* \end{pmatrix} \begin{pmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \mathbf{u}_3 \\ \vdots \\ \mathbf{u}_n \end{pmatrix} = \begin{pmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \mathbf{v}_3 \\ \vdots \\ \mathbf{v}_n \end{pmatrix}$$

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

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

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

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 [6]). The second stage consist of solving two recurrence equations

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

Since blocks D_i and $\Lambda_{i,j}$ in the recurrences (6) are diagonal the solution of $n_2 n_3$ 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 [4]). Each iteration consists of one matrix vector multiplication involving matrix A, one multiplication involving 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 dimensions 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

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

where $b(p)$ denotes time to broadcast a single value 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, for instance, when only the leading

terms are taken into consideration, for the shared memory parallel computer the above function simplifies to

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

Next we analyze the relative speedup S_p and the relative efficiency E_p, where $S_p = \frac{T(1)}{T(p)} \leq p$ and $E_p = \frac{S_p}{p} \leq 1$. Thus the formula for the speedup becomes

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

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 approaches 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 denominators in (8) is significant, in this case the efficiency is much smaller than 1.

4 Experimental Results

In this section we report the results of the experiments executed on Sun Ultra-Enterprise 3000, Cray J-9x and T3E, SGI Origin 2000 computers and on two PC clusters. The code has been implemented in C and the parallelization has been facilitated using the MPI [7] library. In all cases the manufacturer provided MPI kernels have been used. No machine-dependent optimization has been applied to the code itself. 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 and, as verification, the *clock* Unix timer. Results reported by both timers were very close to each other. In all cases we report the best results from multiple runs in interactive and batch modes. In Table 1 we report results obtained on the Sun, the vector-Cray and the SGI computers for $n_1 = n_2 = n_3 = 64, 96, 128, 144, 160$ and for the number of processors p that exactly divides the dimensions of the problem (a temporary limitation of the experimental code). The Sun has 8 processors. On the Cray J-9x and the SGI Origin we could effectively use only up to 16 processors. On the Cray, for larger problems, due to the memory limitation, we could not even use these 16 processors. We report time $T(p)$, speedup S_p (calculated as time on one processor divided by the time on p processors), and efficiency E_p.

A number of observations can be made. First, the proposed implementation, which in a natural way follows the algorithm description, is clearly not appropriate for the vector computer. To be able to achieve a respectable performance on the Cray a vector-oriented implementation would be necessary. Second, for small problems, the proposed approach parallelizes rather well on both shared memory (Sun) and dynamic shared (SMP) machines (SGI). However, as the problem size increases parallel efficiency of the Sun decreases. It can be assumed that this

Table 1. Parallel performance on the SUN Enterprise 3000 superserver, the Cray J-9x vector-computer and the SGI Origin 2000 dynamic shared memory parallel computer

		SUN			Cray J-9x			SGI		
n	p	$T(p)$	S_p	E_p	$T(p)$	S_p	E_p	$T(p)$	S_p	E_p
64	1	2.39			14.07			0.92		
	2	1.16	2.06	1.03	7.32	1.92	0.96	0.46	2.00	1.00
	4	0.60	3.99	1.00	3.87	3.63	0.91	0.23	4.00	1.00
	8	0.31	7.66	0.96	2.21	6.36	0.80	0.12	7.66	0.96
	16				1.86	7.56	0.47	0.09	9.38	0.64
96	1	18.38			44.81			5.56		
	2	9.02	2.04	1.02	23.10	1.93	0.97	2.75	2.02	1.01
	3	6.08	3.02	1.01	16.14	2.77	0.93	1.96	2.83	0.95
	4	4.68	3.93	0.98	12.04	3.72	0.93	1.38	4.02	1.01
	6	3.19	5.76	0.96	8.76	5.11	0.85	0.96	5.67	0.97
	8	2.90	6.34	0.79	6.79	6.59	0.82	0.74	7.51	0.94
	12				5.38	8.32	0.69	0.54	10.29	0.86
	16				5.61	7.98	0.50	0.47	11.82	0.74
128	1	27.67			130.75			10.64		
	2	12.85	2.15	1.08	69.12	1.89	0.95	5.41	1.96	0.98
	4	9.33	2.97	0.74	35.36	3.69	0.92	3.11	3.42	0.86
	8	6.17	4.49	0.56	20.09	6.50	0.81	1.33	8.00	1.00
	16				12.85	10.17	0.64	0.78	13.64	0.85
144	1	70.19			167.55			20.92		
	2	35.21	1.99	1.00	96.23	1.74	0.87	10.64	1.96	0.98
	3	23.79	2.95	0.98	58.32	2.87	0.96	7.05	2.96	0.99
	4	21.52	3.26	0.82	47.37	3.53	0.88	5.57	3.75	0.94
	6	21.45	3.27	0.55	36.55	4.58	0.76	3.55	5.89	0.98
	8	15.39	4.56	0.57	34.76	4.82	0.60	2.67	7.83	0.98
	12				31.82	5.26	0.44	1.84	11.36	0.95
	16							1.46	14.32	0.90
160	1	112.66			223.03			31.85		
	2	46.63	2.42	1.21	116.43	1.87	0.96	14.74	2.16	1.08
	4	24.39	4.62	1.15	61.60	3.77	0.91	7.34	4.33	1.08
	5	28.06	4.01	0.80	50.96	4.65	0.88	6.01	5.29	1.06
	8	21.36	5.27	0.66	36.48	6.83	0.76	3.84	8.29	1.04
	10				32.51	8.43	0.69	2.99	10.65	1.07
	16							2.01	15.84	0.99

is due to the communication overhead which saturates the memory-processor pathways. In addition, the single processor performance follows the same pattern. While for $n = 64$ it takes the Sun twice as long to solve the problem, this ratio increases to almost four times longer for $n = 160$. This observation should also be related to the appearance of super-linear speedup. This effect is visible not only on the Sun, but also, for the largest problem, on the SGI. This effect has a relatively simple explanation. It has been observed many times that, on the RISC based hierarchical memory computers, as the problem size increases their efficiency rapidly decreases (see for instance [1]).

In Table 2 we present the results of our experiments on the Cray T3E and the two PC clusters: the Beowulf cluster of 16 233 MHz PII processors and the Scali cluster of 16 450 MHz PIII processors. The reason for this combination

Table 2. Parallel performance on the Cray T3E and the PC clusters

		Cray T3E			Beowulf cluster			Scali cluster		
n	p	$T(p)$	S_p	E_p	$T(p)$	S_p	E_p	$T(p)$	S_p	E_p
64	1	1.39			2.81			0.90		
	2	0.68	2.04	1.02	1.84	1.52	0.76	0.48	1.88	0.94
	4	0.35	3.97	0.99	1.01	2.78	0.70	0.25	3.60	0.90
	8	0.20	6.95	0.87	0.70	4.01	0.50	0.12	7.50	0.94
	16	0.11	12.63	0.79	0.49	5.73	0.36	0.06	15.00	0.94
96	1	7.46			17.06			5.34		
	2	3.74	1.99	1.00	10.14	1.68	0.84	2.75	1.94	0.96
	3	2.54	2.94	0.98	6.99	2.44	0.81	1.90	2.83	0.94
	4	1.90	3.92	0.98	5.31	3.21	0.80	1.42	3.76	0.93
	6	1.31	5.69	0.95	4.06	4.20	0.70	0.97	5.57	0.93
	8	0.98	7.61	0.95	3.26	5.23	0.65	0.73	7.31	0.91
	12	0.67	11.13	0.93	2.35	7.25	0.60	0.49	10.96	0.91
	16	0.52	14.34	0.90	1.98	8.61	0.54	0.37	14.43	0.89

is that the Cray in the NERSC center has only 256 Mbytes of memory per processor (which is exactly the same amount of memory as we had per node in both clusters) and thus we were able to run on them only the smaller problems. In addition, all three machines represent pure message passing environments.

The results are rather surprising. The Cray is only 3-4 times faster that the 233 MHz PII cluster and slower than the 450 MHz PIII cluster. It should be also added here, that the code on the Beowulf was compiled using the GNU compiler, while the code on the Scali cluster was compiled using the Portland Group compiler and thus the Beowulf results could have been somewhat better if the better quality compiler was used. Observe also that for $n = 96$ the Beowulf cluster has a performance comparable to the Sun (see Table 1). Interestingly, the Scali cluster slightly outperforms the SGI supercomputer. It is a pity that the distributed memory machines did not have more memory per node as it would be very interesting to find out if this relationship holds also for larger problems.

5 Concluding Remarks

In this note we have reported on the parallel performance of a new preconditioner applied to the conjugate gradient method used to solve a sparse linear system arising from a 3D elliptic model problem. We have shown that the code parallelizes well on a number of machines representing shared memory, dynamic shared memory (SMP) and message passing environments. In the near future

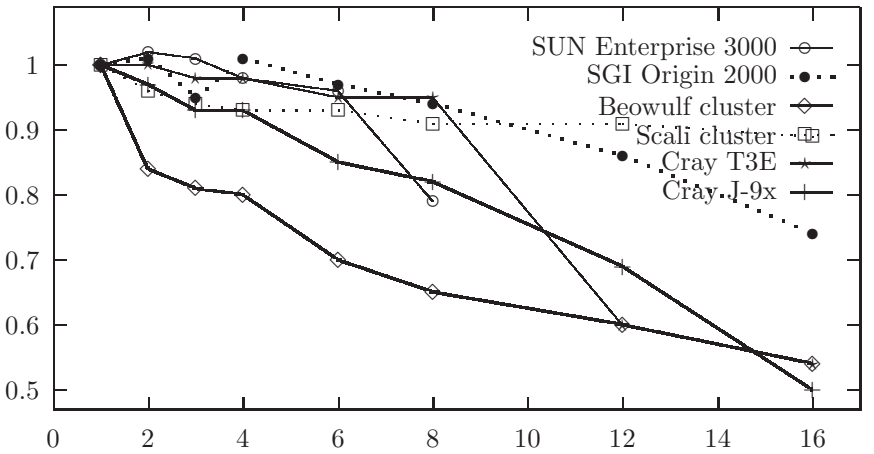

Fig. 1. Parallel efficiency E_p for n=96

we plan, first, to complete the performance studies by running our code on a number of additional machines (e.g. IBM SP2, HP SPP 1000, Compaq Alpha Cluster etc.). Second, we will extend our work to non-uniformly shaped domains, non-uniform discretizations as well as situations when the proposed approach is embedded in a solver for non-linear problems.

Acknowledgments

This research has been supported by Bulgarian NSF Grant I-701/97. Computer time grants from NERSC and MCSR are kindly acknowledged. We would also like to thank Prof. Horst Simon for helping us obtaining accounts on the NERSC machines and to Mr. Anders Liverud from Scali for running the experiments on their cluster.

References

1. I. Bar-On, M. Paprzycki, High Performance Solution of Complex Symmetric Eigenproblem, *Numerical Algorithms*, **18** (1998) 195–208. 540
2. R. H. Chan, T. F. Chan, Circulant preconditioners for elliptic problems, *J. Num. Lin. Alg. Appl.*, **1** (1992) 77–101. 536
3. I. Lirkov, 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. 536, 537
4. I. Lirkov, 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, (1999) 455–463. 536, 538

5. I. Lirkov, S. Margenov, P. S. Vassilevski, Circulant block-factorization preconditioners for elliptic problems, *Computing*, **53** 1 (1994) 59–74. 536
6. C. Van Loan, *Computational frameworks for the fast Fourier transform*, SIAM, Philadelphia, 1992. 538
7. M. Snir, St. Otto, St. Huss-Lederman, D. Walker, J. Dongara, *MPI: The Complete Reference*, Scientific and engineering computation series, The MIT Press, Cambridge, Massachusetts, 1997, Second printing. 539