Proceedings of the 10th IASTED International Conference
PARALLEL AND DISTRIBUTED COMPUTING AND SYSTEMS
October 28-31, 1998, Las Vegas, Nevada, USA

PARALLEL SOLUTION OF 2D ELLIPTIC PDE'S ON SILICON GRAPHICS SUPERCOMPUTERS

IVAN LIRKOV
CLPP, BAS
Acad. G. Bonchev St. Block 25A
1113 Sofia, Bulgaria

SVETOZAR MARGENOV
CLPP, BAS
Acad. G. Bonchev St. Block 25A
1113 Sofia, Bulgaria

MARCIN PAPRZYCKI
Dept. of Computer Science and Stat.
University of Southern Mississippi
Hattiesburg, MS 39406, USA

ABSTRACT

It was recently shown that block-circulant preconditioners are very efficient when applied to a conjugate gradient method used to solve a sparse linear system arising from discretizations of 2D elliptic partial differential equations. In addition, the initial experimental data, collected for small and medium size systems solved on parallel computers with limited number of processors, indicated potential for a good parallel efficiency. The aim of this note is to compare the performance of the 2D elliptic solver based on a conjugate gradient iterative solver with block-circulant preconditioner across four parallel computers from the Silicon Graphics Inc.: SGI Power Challenge 8000, SGI Power Challenge 10000, SGI Origin2000 running at 195MHz and SGI Origin 2000 running at 250 MHz.

KEYWORDS

Partial differential equations, conjugate gradient method, block-circulant preconditioning, parallel performance

INTRODUCTION

Consider a numerical solution to an elliptic partial differential equation (PDE). A finite difference (as well as a finite element) discretization transforms this continuous problem to a discrete one. The discrete problem can be represented by a linear system $Ax = b$, where A is a symmetric positive definite matrix (and, if an appropriate ordering of variables is applied, this matrix is block tridiagonal). Conjugate gradient type methods are recognized as the most cost-effective way to solve large problems of this type. To accelerate the convergence of the iterative solver a preconditioner matrix C is combined with the conjugate gradient algorithm. The theory of iterative solvers shows that C is considered a good preconditioner if it minimizes the condition number of the matrix $C^{-1}A$ and allows an efficient computation of $C^{-1}v$ for a given vector v.

Recently, a new class of precoditioners has been introduced by Lirkov, Margenov and Vassilevski in [1]. The *circulant block-factorization* (CBF) preconditioner is based on a circulant approximation of the blocks of the matrix A (considered in its block-tridiagonal form). This preconditioner belongs to the same class as the well-known *block-circulant* (BC) preconditioner introduced by R.H. Chan and T.F. Chan in [2]. Both preconditioners are based on block-wise average of the coefficients of A and are motivated by the ability to exploit the fast inversion of circulant blocks by Fast Fourier Transform (see [3] for more details). In addition, it has been shown that preconditioners of this type result in a favorable clustering of eigenvalues of the preconditioned systems (see for instance [3, 4, 5]). It should be mentioned, that for the model Poisson problem the condition number of the preconditioned system is $O(\sqrt{n})$, where n is the number of unknowns, and thus is the same as for the standard *ILU* based preconditioners. The advantage of the circulant preconditioners is their parallel efficiency.

Results of initial experiments indicated that the CBF preconditioner leads to a slightly faster convergence of the iterative process than the BC preconditioner (for more details see [1, 6]). It has also been shown that when the parallel performance characteristics of both preconditioners are considered the CBF outperforms the BC preconditioner on a distributed memory (network of transputers) and on shared memory parallel computers (see [7, 8] for the respective experimental results). The results reported in [7] and [8] have been for very small problems and for a limited number of processors. It was caused by first, the available hardware (small network of transputers with a limited local memory) and, second, some problems encountered with the PVM-based implementation of the algorithms [8]. Since then the algorithm has been re-implemented using the MPI primitives and the earlier problems have been eliminated.

The aim of this note is to summarize the performance characteristics of the CBF preconditioner for relatively large problems on four high performance parallel computers from the Silicon Graphics Inc (two bus-based shared-memory computers: SGI Power Challenge 8000 and Power Challenge 10000 and two single-system image Origin2000 computers which differed in their processor speed). Since the results collected thus far indicated clearly

that the CBF preconditioner outperforms the BC preconditioner we have decided to concentrate our attention on the CBF preconditioner only.

The remaining parts of this note are organized as follows. In the next section we briefly introduce the basic idea behind the circulant preconditioner. The parallel complexity of the proposed algorithm is summarized next and followed by the presentation and discussion of experimental results. We conclude the paper with a short summary and a description of future research.

CIRCULANT PRECONDITIONERS

Consider a 2D second order elliptic PDE on the unit square (0, 1) × (0, 1) with homogenous Dirichlet boundary conditions. Let the domain be triangulated by a standard rectangular grid with n grid points in each direction and let the usual 5-point stencil finite-difference approximation be applied to the problem. This discretization leads to a linear system $Ax = b$, where A is symmetric positive definite and, if the gridpoints are ordered along the y-direction first, the matrix A is block-tridiagonal and can be represented as:

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

Matrix C is circulant if $(C_{k,j}) = (C_{(j-k) mod m})$, where C is an $m \times m$ matrix. The CBF preconditioner is defined by

$$C_{CBF} = tridiag(-C_{i,i-1}, C_{i,i}, -C_{i,i+1}),\ i = 1,2,\ldots, n$$

where $C_{i,j} = circulant(A_{i,j})$ is a proper circulant approximation of the corresponding block $A_{i,j}$ (for more details see [1]).

PARALLEL COMPLEXITY

It has been shown that when an iterative method is considered (or when multiple iterative methods of the same asymptotic convergence are compared) the best model of analyzing the parallel complexity is to consider the cost per iteration. The cost of the iteration consists of the cost of arithmetical operations (we will assume that the same number of operations is performed on each processor) and the cost of data communication between processors. Let us assume that t_a is the average time to perform an arithmetic operation on a single processor while $T_{com} = t_s + M t_c$ is the communication time of transmitting M elements (it consists of the startup time t_s and the communication time of transmitting one element t_c). Then it can be shown (see [1, 8] for the detailed derivation) that, after a number of simplifying conditions, the total parallel complexity of one iteration of the CBF preconditioned conjugate gradient method can be represented as:

$$T_{PCG}^{CBF} \approx 2pt_s + 2(1-\frac{1}{p})\frac{n^2}{p}t_c + (31+10\log n)\frac{n^2}{p}t_a,$$

where p is the number of processors. It can be also shown that, if S_p denotes speedup and E_p denotes efficiency then as $n \to \infty$, $S_p = p$ and $E_p = 1$ and the algorithm is asymptotically optimal. It should be also observed, that for small n the startup time t_s is usually much larger than t_a and the efficiency is much smaller than 1.

EXPERIMENTAL RESULTS

We have performed our experiments on four parallel computers produced by the Silicon Graphics Inc. These machines are a part of the National Center for Supercomputing Applications (NCSA) in Urbana. Two bus-based shared memory systems were: the 16-processor Power Challenge 8000 (PC8) based on the MIPS R8000 processors running at 90 MHz with a theoretical peak performance of 360 Mflops, with 4 Gbytes of memory and the 16-processor Power Challenge 10000 (PC10) based on MIPS R10000 processors running at 195 MHz with a theoretical peak performance of 390 Mflops, with 4 Gbytes of memory. The remaining two machines were single-system image (dynamic shared memory) Origin2000 computers. First (O1) had 64 processors and 8 Gbytes of memory and was based on the 195 MHz MIPS R10000 processors (the same technology as for the PC10). Second (O2) had 64 processors, 16 Gbytes of memory and was based on the newest 250 MHz MIPS R10000 processors capable of theoretical peak performance of 500 Mflops (more details about these machines can be found at the NCSA WWW site http://www.ncsa.uiuc.edu, or at the manufacturer WWW site http://www.sgi.com).

The algorithm was implemented in C using the MPI communication library to facilitate the data exchange between processors. In all experiments exactly the same code was used. This means that on the shared memory machines the MPI-based communication was used. For all machines the most aggressive of available compiler optimizations were turned on and the manufacturer provided libraries of optimized MPI kernels have been linked with the code. The MPI provided timer was used to measure the execution times. Each result presented here is either the best time obtained from multiple runs on a system with varying workload or was obtained in a dedicated mode (when the code was the only one that was running on the system).

The Results from the Power Challenge 8000 and 10000

The first series of experiments was performed on the Power Challenge 8000 (PC8) shared memory computer. Table 1 contains the execution times and the speedup for n = 210, 840 and for p = 1, 2, …, 8 processors. Since the current implementation requires that the problem size be divisible by the number of processors, for n = 210 results for 4 and 8 processors are not available.

TABLE 1. PERFORMANCE ON THE PC8

	$n = 210$		$n = 840$	
p	time	speedup	time	speedup
1	0.43	1	8.64	1
2	0.22	1.92	4.49	1.92
3	0.16	2.77	2.88	3.00
4			2.43	3.55
5	0.091	4.75	1.99	4.34
6	0.097	4.46	1.53	5.64
7	0.092	4.70	1.47	5.87
8			1.26	6.85

While the performance for the smaller problem was relatively unimpressive problem for more than 5 processors, this just indicates that the problem was much too small for the speed of the machine. For the larger system an efficiency of approximately 85% was achieved for 8 processors, which is a rather good result.

The second series of experiments involved the Power Challenge 10000 machine (PC10). The R10000 processor runs at a much higher MHz rate than the R8000, but the performance improvements were primarily focused on the integer arithmetic and thus its practical Mflop peak was only slightly higher [9]. In Table 2 we present the execution time and the speedup of the code for $n = 420$, 840 for $p = 1, 2, \ldots, 8$ processors (since the parallel performance for $n = 210$ was even worse than in the case of PC8 we decided to skip it and report the results for $n = 420$ instead).

TABLE 2. PERFORMANCE ON THE PC10

	$n = 420$		$n = 840$	
p	time	speedup	time	speedup
1	1.26	1	4.75	1
2	0.72	1.75	2.55	1.86
3	0.51	2.44	1.73	2.74
4	0.36	3.46	1.35	3.51
5	0.32	3.84	1.09	4.35
6	0.26	4.74	0.95	5.00
7	0.27	4.51	0.84	5.65
8			0.76	6.26

The performance for the smaller problem (even though it is now twice the size of the problem reported for the PC8 above) is very unsatisfactory for more than 4-5 processors. The efficiency for the larger problem reaches 78% for 8 processors. This is less than the efficiency obtained on the PC8 and likely can be explained by the fact that the processors of the PC10 are faster and thus for the same problem size there is not enough work to keep them busy.

When the ratio of execution times of the larger problem on the two PC machines is considered it can be observed that PC10 is approximately 1.81 times faster for a single processor while only 1.65 times faster on eight processors. This shows that first, the improved integer performance plays an important role in the case of our code. Second, since the performance ratio decreases as the number of processors increases, it seems to indicate again that the processor to memory communication on the PC10 is not fast enough to keep up with the increased processor speed.

The Results from the Origin2000

The next series of experiments was performed on the SGI Origin2000 single-system image machine (O1). It is based on the same MIPS R10000 processors as the PC10, but it is designed as a dynamic shared memory machine. In Table 3 we depict the execution times and the speedup of the code for $n = 420$, 840, for $p = 1, \ldots, 8$ processors.

TABLE 3. PERFORMANCE ON THE O1

	$n = 420$		$n = 840$	
P	time	Speedup	time	speedup
1	0.92	1	4.02	1
2	0.48	1.90	2.25	1.78
3	0.32	2.83	1.48	2.70
4	0.24	3.81	1.11	3.61
5	0.19	4.75	0.87	4.60
6	0.16	5.46	0.72	5.52
7	0.14	6.37	0.62	6.48
8			0.54	7.40

The results, especially for the smaller problem, are rather surprising. While based on the same processor as the PC10, the Origin2000 obtains substantially faster execution times on single and multiple processors and a much better speedup. It can be speculated that since O1 is a newer machine some improvements in the architecture (e.g. cache memory and interconnection network) are responsible for these improvements (unfortunately on the WWW pages at NCSA we could not find any data to support this hypothesis). For the larger problem a very good efficiency of 92% is obtained on 8 processors.

The final series of experiments was completed on the newest Origin2000 machine (O2) which became available to the NCSA users in July 1998. In comparison with O1 it is based on a faster version of the MIPS R10000 processor. Table 4 summarizes the execution times and speedup of the code for $n = 420$, 840, for $p = 1, \ldots, 8$ processors.

TABLE 4. PERFORMANCE ON THE O2

	$n = 420$		$n = 840$	
p	time	Speedup	time	speedup
1	0.74	1	3.23	1
2	0.40	1.85	1.90	1.70
3	0.27	2.72	1.26	2.56
4	0.21	3.57	0.93	3.47
5	0.16	4.53	0.73	4.40
6	0.14	5.28	0.62	5.23
7	0.11	6.37	0.53	6.00
8			0.47	6.78

The results are quite interesting. First, for the smaller problem, the overall performance is quite similar to the older machine. While the overall execution time is reduced the speedup is only minimally smaller and for 7 processors is almost exactly the same, resulting in efficiency of about 91%. The situation changes for the larger problem. Here, while the time is reduced (on a single processor the new machine is 1.24 times faster and on 8 processors it is 1.14 times faster) the 8-processor efficiency decreases from 92% on O1 to 84% on the new machine. This seems to indicate that while the processor speed has been improved, the interconnection network was not upgraded to provide the data fast enough to match the speed of the faster processors.

Comparisons between machines

The latter observation prompted us to look into the relationships between the performance of the three machines based on the MIPS R10000 processor. To obtain a more detailed picture we have decided to increase the problem size and the number of processors used. Table 5 reports the execution times, the speedup and the time ratio between the PC10 and O1 for a problem of size $n = 1260$ and 1,2,..., 7, 9, 10, 12, 14, 15 processors. As previously, results for 8, 11, 13 processors were not available as these are not divisors of the problem size.

TABLE 5. PERFORMANCE OF PC10 vs. O1

	PC1000		O1		
p	time	speedup	time	speedup	ratio
1	11.23	1	9.71	1	1.16
2	5.99	1.86	5.50	1.76	1.09
3	4.11	2.74	3.69	2.63	1.11
4	3.17	3.51	2.77	3.49	1.14
5	2.61	4.35	2.21	4.39	1.18
6	2.31	5.00	1.83	5.28	1.26
7	2.09	5.65	1.57	6.18	1.33
9	1.82	6.25	1.21	7.97	1.50
10	1.75	7.19	1.11	8.71	1.57
12	1.57	8.06	0.92	10.53	1.70
14	1.48	8.57	0.80	12.09	1.84
15	1.43	8.37	0.75	12.92	1.90

The results are in agreement with our earlier observations. The technology of the Origin2000 is definitely superior to that of the PC 10000 both in terms of single processor and multiprocessor performance. Even though both machines are based on the same processor, the Origin is approximately 1.16 times faster on a single processor. The time ratio decreases slightly for 2 and 3 processors. However, as the number of processors increases, the performance of the Origin improves to 1.9 times faster than the PC 10000. This affects also other performance measures. For the PC10 the speedup on 15 processors reaches only 8.37 (the efficiency of 56%) and the speedup curve flattens from about 10 processors. On the O1, on the other hand, the speedup reaches 12.92 (the efficiency of 86%) and the speedup curve flattens only from about 14-15 processors.

In Table 6 we present similar results comparing the performance between the two Origin2000 machines.

TABLE 6. PERFORMANCE OF O1 vs. O2

	O1 (195MHz)		O2 (250MHz)		
p	time	speedup	time	speedup	ratio
1	9.71	1	7.71	1	1.26
2	5.5	1.76	4.57	1.68	1.20
3	3.69	2.63	3.09	2.49	1.19
4	2.77	3.49	2.33	3.31	1.19
5	2.21	4.39	1.87	4.12	1.18
6	1.83	5.28	1.55	4.98	1.18
7	1.57	6.18	1.33	5.80	1.18
9	1.21	7.97	1.05	7.35	1.16
10	1.11	8.71	0.91	8.39	1.21
12	0.92	10.53	0.82	9.35	1.12
14	0.80	12.09	0.71	10.81	1.12
15	0.75	12.92	0.66	11.66	1.13

The results are rather informatory. The new machine has a MHz rate 1.28 times faster and its theoretical peak performance (measured in Mflops) is also 1.28 times higher. These facts do not seem to manifest themselves in our data. On a single processor O2 is about 1.26 times faster than O1. However, as the number of processors increases the ratio drops considerably and for more than 12 processors O2 is only 1.12 times faster than O1. This confirms our earlier hypothesis that while the processor speed has been increased the network was not improved to provide appropriate throughput. This fact has also an immediate effect on the overall performance metrics. The speedup of the code on 15 processors drops from 12.92 on O2 to 11.66 on the new machine (efficiency drops from 86% to 77%).

Performance for large problems

Finally, to examine how the SGI machines will behave when the memory hierarchy and the interconnection network are being tested by a movement of large volumes of data we have run a series of experiments for a rather

large problem (n = 2520). Table 7 summarizes the execution times on all four machines for p = 1,2,..., 10, 12, 14, 15 processors. As previously, results for 11 and 13 processors were not available as these are not divisors of the problem size. It should be stressed that for this problem, all data sets on PC10, O1 and O2 were collected in a dedicated mode.

TABLE 7. PERFORMANCE FOR A LARGE PROBLEM

p	PC 8	PC 10	O1	O2
1	84.53	47.54	42.65	33.10
2	44.68	24.76	23.29	19.15
3	30.15	17.10	15.71	12.94
4	22.79	13.23	11.75	9.85
5	18.94	11.07	9.58	8.01
6	16.01	9.67	7.98	6.66
7	14.65	8.74	6.97	5.79
8	12.92	7.97	6.04	5.12
9	12.20	7.50	5.36	4.65
10	11.41	7.24	4.83	4.07
12	11.26	6.73	4.02	3.40
14	11.16	6.37	3.43	2.91
15	10.46	6.29	3.24	2.77

The performance ratio between the PC8 and PC10 remains very similar to that reported earlier for a small problem. The single-processor ratio decreases slightly from 1.81 to 1.77 while the 15-processor ratio is approximately 1.66. This indicates that the two machines behave similarly independently of the problem size.

As previously, the difference between PC10 and O1 manifests itself mainly for the large number of processors. While for a single processor the performance ratio decreases from 1.16 to 1.11, for 15 processors the ratio reaches 1.94.

The performance ratio between the two Origin2000 computers is also rather similar to the earlier results. On a single processor the new machine takes a full advantage of the improved processor speed and becomes 1.288 times faster. However, as the number of processor increases the gains disappear again. For 15 processors the performance ratio decreases to only 1.16.

Finally, in Figure 1 we represent the speedup of the code on all four SGI machines for the largest problem studied (n = 2520). These results match times depicted in Table 7. To illustrate the performance we also represent the linear speedup. Since 11 and 13 processors had to be omitted from our experiments, the line representing the linear speedup is no longer straight.

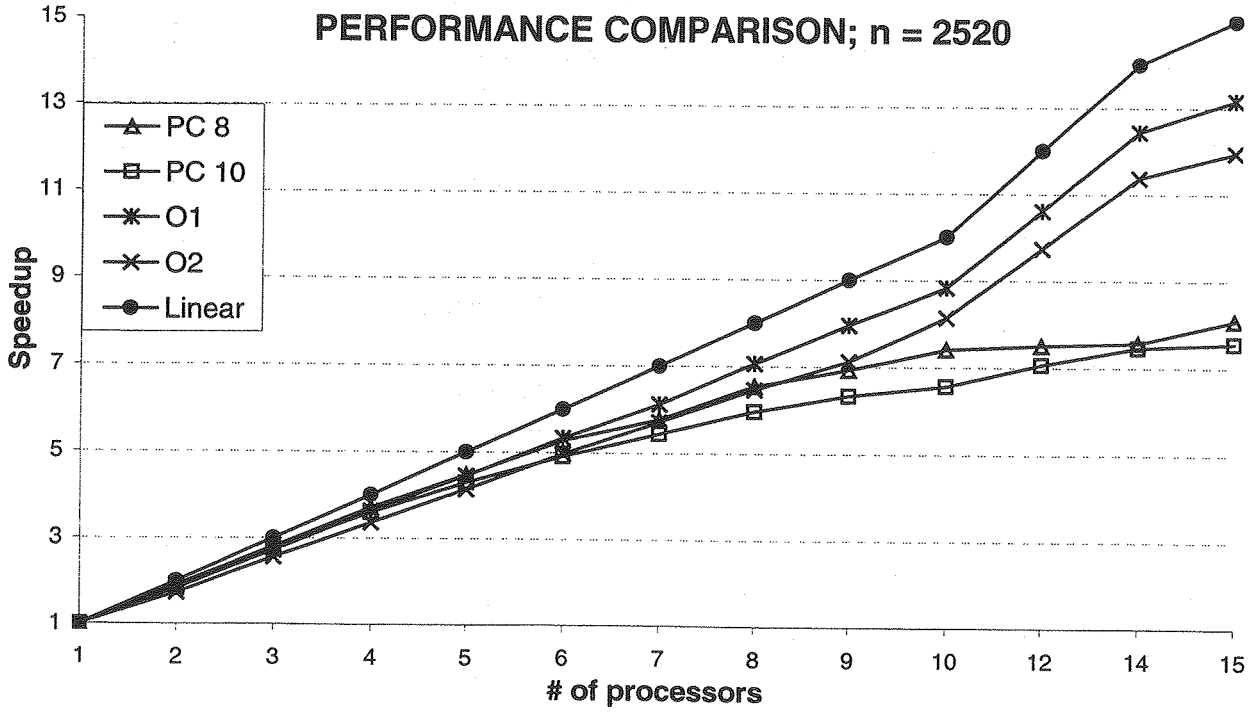

FIGURE 1. PERFORMANCE COMPARISON BETWEEN THE FOUR SGI COMPUTERS.

The results match our earlier observations. For more than 8 processors the speedup curve on the two shared memory machines flattens indicating that the bus has been saturated and further performance gains cannot be expected. This effect seems to be relatively independent of the problem size (see also Table 5 above) and its further increase cannot be expected to lead to substantial performance gains either. The two Origin2000 machines perform relatively well. The older machine showing a slightly better balance between the processor speed and the network throughput. This results in a better speedup (reaching 13.16) and efficiency (approximately 87%) while the maximum speedup obtained on O2 is 11.94 (efficiency only 79%).

CONCLUDING REMARKS

In this paper we have studied the performance characteristics of a parallel solver for the elliptic partial differential equations. The proposed solver is based on finite differences and the linear algebra consists of a circulant block-factorization based preconditioner applied to a conjugate gradient solver for a block tridiagonal linear system. The performance was studied on two bus-based shared memory computers and two single-system image parallel machines. Our results indicate that, for large enough problems, the proposed solver parallelizes well on all machines and achieves high speedup and a relatively good efficiency.

The experimental data collected in our experiments can be also used to compare and evaluate the performance characteristics of the four parallel architectures. It is clear that the two bus-based shared memory machines suffer from communication congestion when a large number of processors is used. This effect seems to be relatively independent of the problem size. We have also found the Origin2000 architecture seems to match quite well the requirements of our algorithm. At the same time, the new faster Origin2000 is incapable of taking full advantage of the improved processor power. Its communication network has not been improved to the point where it will match the new processor speed. Obviously, the new machine still remains faster overall indicating that for a problem with a more localized data access patterns it should be able to outperform the older architecture to a larger extent.

Our work thus far was mostly with a 2D model problem. The next step will be to develop and experiment with an elliptic solver for a 3D model problems and, later, to apply the newly developed solver to some realistic applications. We plan to pursue this research avenue in the near future.

ACKNOWLEDGEMNETS

Computer time grant (Project qtp) from NCSA in Urbana is kindly acknowledged. We would also like to express our gratitude to Ray Seyfarth and Julian Sanchez who helped in porting the code to different machines.

REFERENCES

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

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

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

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

[5] T. Huckle, Some aspects of circulant preconditioners, *SIAM J. Sci. Comp.*, 14, 1993, 531-541.

[6] I. Lirkov, S. Margenov and L. Ziakatanov, Circulant block-factorization preconditioning for anisotropic elliptic problems, *UCLA CAM Report 95-39*, 1995.

[7] I. Lirkov and S. Margenov, Parallel complexity of conjugate gradient method with circulant preconditioners, in: R. Vollmar, W. Erhard and V. Jossifov, eds., *Proceedings of the PARCELLA'96* (Akademie Verlag, Berlin, 1996), 279-286.

[8] I. Lirkov, S. Margenov, R. Owens and M. Paprzycki, A shared-memory parallel implementation of block-circulant preconditioners, in: M. Griebel, O. P. Illiev, S. Margenov and P. S. Vassilevski eds. *Proceedings of the First Workshop on "Large-Scale Scientific Computations,"* (VIEWEG, Braunschweig, 1997), 319-327.

[9] J. Foster, Evolution of MIPS RISC Microprocessor Architecture, *Journal of Computing in Small Colleges*, 2(4), 1997, 215-229