BENCHMARKING PERFORMANCE OF PARALLEL COMPUTERS USING A 2D ELLIPTIC SOLVER

IVAN LIRKOV AND SVETOZAR MARGENOV
Central Laboratory for Parallel Processing, Bulgarian Academy of Sciences
Acad. G. Bonchev Str., Bl.25A, 1113 Sofia, Bulgaria

MARCIN PAPRZYCKI
Department of Computer Science and Statistics, University of Southern Mississippi, Hattiesburg, Mississippi, 39406-5106, USA

It was recently shown that block-circulant preconditioners applied to a conjugate gradient method used to solve structured sparse linear systems arising from 2D elliptic problems have very good numerical properties and a potential for good parallel efficiency. The aim of the presentation is to summarize and compare their parallel performance across a number of modern parallel computers: SGI Power Challenge 10000, SGI/Cray Origin 2000, HP-Convex SPP-2000, IBM SP-2, Cray T3E, a network of SUN workstations and a Beowulf cluster of PC's.

1 Introduction

When a second order elliptic partial differential equation (PDE) on a square is discretized using one of the standard discretization techniques the resulting discrete problem can be represented as a linear system $Ax = b$. Here matrix A is large block tridiagonal and symmetric positive definite. Typically, to solve this linear system a preconditioned conjugate gradient method is used. Recently, a modification to the well known *block-circulant* (BC) preconditioner was proposed by Lirkov, Margenov and Vassilevski (see [?] for more details). The *circulant block-factorization* (CBF) preconditioner is based on a circulant approximation to the blocks of matrix A and uses FFT's for fast inversion of circulant blocks.

Initial results indicated that the CBF preconditioner has somewhat better numerical properties than the standard BC preconditioner and its parallel performance is also slightly better. These results were collected on distributed and shared memory parallel computers (see [?,?] for more details). Due to the hardware limitations and problems with the PVM-based implementation of the new algorithm these results were collected for small and medium size problems only. After the code was re-implemented using MPI primitives and with special attention paid to the data communication we were able to experiment with the larger problems. The initial data from these experiments on four computers from Silicon Graphics Inc. has been reported in [?].

The aim of this note is to compare the performance of a wide range of high performance computers while running the same code. The machines are: SGI Power Challenge 10000, SGI/Cray Origin 2000, HP-Convex SPP-2000, IBM SP-2, Cray T3E, a network of SUN workstations and a Beowulf cluster of PC's. The remaining part of the paper is organized as follows: in Section 2 the basic information about the circulant preconditioner and its parallel complexity is presented. Section 3 contains the summary of the experimental data. In the final section we summarize our findings and sketch the future research direction.

2 Circulant preconditioners and parallel complexity

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

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

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

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

where $C_{i,j} = Circulant(A_{i,j})$ is a proper circulant approximation of the corresponding block $A_{i,j}$ (for more details see [?]). The relative condition number of the preconditioned system for a model Poisson problem for CBF preconditioners is estimated by $O(\sqrt{n})$, i.e., the same result as for certain incomplete LU factorization type preconditioners holds. The advantage of the CBF preconditioners is their parallel efficiency.

It has been shown that when an iterative method is considered the best model of analyzing the parallel complexity is to consider the cost per iteration. The cost of one preconditioned conjugate gradient 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 $T_{com} = t_s + Mt_c$ 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).

The general estimate for the total parallel complexity of one iteration of the CBF preconditioned conjugate gradient method on distributed or shared memory parallel system with P processors is derived in [?]. We have

$$T_{PCG}^{CBF}(P) \approx (31 + 10\log n)\frac{n^2}{P}t_a + 2Gthr(\frac{n^2}{P}, P) + 2Gthr(P, P) + 2Brdc(P),$$

where $Gthr(M, P)$ is time for gathering P data packets, each packet with M/P words and $Brdc(P)$ is time for broadcasting a number from one processor to all others. For distributed memory architecture models: ring, square grid and hypercube the above communication times are given by:

	$Brdc(P)$	$Gthr(M, P)$
ring	$(t_s + t_c)\frac{P}{2}$	$P\,t_s + M\,t_c$
square grid	$(t_s + t_c)\sqrt{P}$	$2\sqrt{P}\,t_s + M(1 + \frac{1}{\sqrt{P}})\,t_c$
hypercube	$(t_s + t_c)\log P$	$\log P\,t_s + M(1 - \frac{1}{P})\,t_c$

It can be shown that, after a number of simplifying assumptions, the total parallel complexity of one iteration of the CBF preconditioned conjugate gradient method on a shared memory parallel system is (for details see [?]):

$$T_{PCG}^{CBF}(P) \approx 2Pt_s + 2(1 - \frac{1}{P})\frac{n^2}{P}t_c + (31 + 10\log n)\frac{n^2}{P}t_a.$$

It can be also shown that, if S_P denotes speedup and E_P denotes efficiency then $\lim_{n\to\infty} S_P = P$ and $\lim_{n\to\infty} E_P = 1$, and the algorithm is asymptotically optimal.

3 Experimental results

3.1 Experimental setup

The experimental data was collected on nine different parallel computers: SGI Power Challenge 8000, SGI Power Challenge 10000, SGI/Cray Origin 2000, HP-Convex SPP-2000 (Exemplar), IBM SP-2, Cray T3E, a network of SUN workstations and a Beowulf cluster of PC's. Table **??** summarizes the hardware specifications and the performance characteristics of these machines. Since the results from earlier experiments on the four Silicon Graphics machines have been summarized in [?] we will omit the timings from the older

Table 1. Machine characteristics

Machine	memory		processor	MHz	MFlops
PC 8000	shared	16Gb	MIPS R8000	90	360
PC 10000	shared	16Gb	MIPS R10000	195	390
Origin 2000	dynamic	12Gb	MIPS 10000	195	390
	shared	32Gb		250	500
SPP-2000 Exemplar	dynamic shared	4Gb	PA-RISC 8000	180	720
IBM SP-2	distributed	1Gb	IBM Power2 Thin Node	160	360
SUN network	distributed	128Mb	UltraSPARC	167	330
Beowulf	distributed	256Mb	Pentium II	233	88
Cray T3E	distributed	128Mb	DEC Alpha 21164	400	500

Power Challenge 8000 and the slower Origin 2000. For the purpose of completeness of comparison we will, however, report the results from the remaining two SGI machines.

All experiments have been run using exactly the same code implemented in C. MPI library was used to facilitate parallelism. Even on a shared memory machine (SGI PC 10000) the MPI based parallelism was used. The manufacturer provided optimized MPI primitives were used (in case of the Beowulf cluster the most aggressive *mpirun* options have been turned on). In all cases the most aggressive manufacturer provided compiler optimizations have been turned on as the only form of optimizing the code for the architecture. Each result reported here is either obtained in benchmarking mode on an empty machine, or is the best result obtained from multiple experiments on a machine with varying loads. Timings were obtained using the MPI's provided timer.

3.2 Results for $n = 840$

The first series of experiments was run for a moderately large problem of size $n = 840$. In Table **??** we present the times for the execution of the program for $P = 1, 2, \ldots, 8$ processors.

The results are rather surprising. A single processor performance of the slowest machine (as far as the MFlop rate is concerned), the Beowulf cluster, is better than that of the Sun Workstation delivering the result in about 2/3 of

Table 2. Execution times, speedup, efficiency and MFlops rate for $n = 840$

P	PC 10000	Origin	Exemplar	SP-2	NOW	Beowulf	Cray
1	4.75	3.23	5.51	8.75	21.03	14.27	6.52
2	2.55	1.90	2.62	4.41	10.91	8.19	3.20
3	1.73	1.25	1.90	2.99	7.37	7.24	2.13
4	1.35	0.93	1.41	2.18	5.56	4.46	1.60
5	1.09	0.73	1.14	1.74	4.49	3.47	1.28
6	0.95	0.62	0.92	1.44	3.76	2.82	1.07
7	0.84	0.53	0.84	1.25	3.27	2.49	0.92
8	0.76	0.47	0.71	1.10	2.92	2.24	0.81
Speedup on 8 processors							
8	6.25	6.79	7.76	7.95	7.20	6.37	8.10
Efficiency on 8 processors							
8	0.7812	0.8590	0.9701	0.9943	0.9003	0.7963	1.0124
MFlop rate on 1 and 8 processors							
1	44.71	65.75	38.55	24.27	10.10	14.88	31.07
8	279.45	451.88	299.13	191.34	72.73	94.82	251.61

the time. This can be explained by the fact that the Sun Workstations in the NOW cluster at UC Berkeley are equipped only with 128 MBytes of memory and the system has to swap the data in and out of the disk (the Beowulf machines have 256 MBytes of memory). The 8-processor performance of the two machines is approximately the same. This suggests that the network on the Beowulf is not fast enough and this affects the performance.

Unexpectedly, the single processor performance of the Cray T3E is only 1/2 of that of the Origin; even though both machines have the same theoretical peak performance. This is likely to be related to the lack of sufficient memory on the Cray. This explanation is further supported by the superlinear speedup obtained on the Cray. This is usually an invitation of a memory-related bottleneck, that is removed as the data is distributed among a larger number of processors. This is rather interesting when considering the fact that such an effect has not been observed on the NOW.

The remaining three distributed memory machines cannot match the performance of the other three computers. This is particularly surprising for the IBM's SP-2 which has the single node theoretical peak performance only slightly slower than the PC 10000, but is almost twice slower on a single node. It should be observed that the Exemplar, which has the highest theoretical MFlop rate, is slower than both of the SGI machines. This can be attributed

to the fact that it has only a one-way cache, while the SGI machines have two-way cache memories [?].

The 8-processor performance is relatively good on all machines but the Beowulf and the PC 10000. In cases of these two machines the network/bus performance which is not able to match the processor speed is to blame for the smaller speedup/efficiency. Out of the remaining machines, the efficiency of the Origin is the lowest. This can be related to the fact that the new Origin has a much faster processor, but its network was not upgraded to match this additional computational power (see [?] for additional data and the comparison of the performance of the two Origin machines). The NOW, the Exemplar and the SP-2 deliver rather good parallel performance measured in terms of speedup and efficiency (efficiency of 90%, 97% and 99% respectively). This latter fact, compared with the timing data presented above and the theoretical peak performance summarized in Table 1 suggests once more that both speedup and efficiency can be not only misleading when a performance of a given algorithm on a given machine is concerned [?], but can be also rather misleading when a comparison between machines is undertaken. Here, the three machines that underutilize their potential computational power the most are reported as the most efficient.

3.3 Results for $n = 1260$

The second series of experiments has been executed for a large problem of size $n = 1260$. Table **??** depicts the execution times for the six computers for $P = 1, 2, \ldots, 15$ processors. We were not able to fit the problem into 1 and 2 processors on the Cray (which further supports our earlier discussion of its performance) and thus the results are not reported. Since the code represents only a model problem it requires that the number of processors divides the problem size exactly. Thus the experiments for $P = 8, 11, 13$ processors have been omitted. The 'xxxx' symbol denotes that given data is not applicable.

The results resemble these obtained for $n = 840$. We observe that, also on the Sun Workstation, the 128 MBytes of memory are not enough to fit our problem. Not only it has to swap data to the disk, but it has to do so often that its performance becomes reduced approximately sixfold (when the MFlop rate of about 10 reported in Table 2 is compared to the 1.69 above).

The differences in single processor performance between the remaining machines become more pronounced as far as the wall-clock time is concerned. This suggests that the increase in problem size does not help the Exemplar to overcome the one-way cache problem. Interestingly, when the MFlop rates are compared with these reported in Table 2 above the Beowulf and SP-2

Table 3. Execution times, speedup, efficiency and MFlops rate for $n = 1260$

P	PC 10000	Origin	Exemplar	SP-2	NOW	Beowulf
1	11.23	7.71	12.30	22.53	319.29	37.97
2	5.99	4.57	6.45	11.96	36.12	22.07
3	4.11	3.09	4.34	8.14	18.84	14.74
4	3.18	2.33	3.18	5.54	14.95	11.12
5	2.61	1.87	2.61	4.97	11.74	8.82
6	2.31	1.55	2.31	4.10	9.67	7.45
7	2.09	1.33	1.88	3.24	8.36	6.51
9	1.82	1.05	1.54	3.76	6.64	5.14
10	1.75	0.92	1.34	2.74	5.90	5.71
12	1.57	0.82	1.17	1.91	4.96	3.91
14	1.48	0.71	0.99	2.17	4.46	4.72
15	1.43	0.66	0.98	1.58	4.02	3.57
Speedup on 15 processors						
15	7.85	11.66	12.55	14.26	xxxx	10.63
Efficiency on 15 processors						
15	0.5235	0.7788	0.8367	0.9506	xxxx	0.7091
MFlop rate on 1 and 15 processors						
1	47.92	69.81	43.76	23.89	1.69	14.17
15	376.36	815.45	549.18	340.63	133.88	150.76

report a slight performance reduction, while the remaining machines report a performance increase of 3-5 MFlops.

The parallel performance is similar to that reported for the smaller system. It should be observed that for 15 processors the performance of the shared memory PC 10000 is visibly affected by the communication overhead (its efficiency is only about 52%).

3.4 Results for $n = 2520$

The final set of results reported here is for a rather large system of size $n = 2520$. In Table **??** we depict the results for above systems but the NOW, the Beowulf and the SP-2. In case of NOW already for the system of size 1260 we observed that there is not enough memory. Similar effects have been observed on the Beowulf cluster for up to 15 processors. Finally, on the SP-2 we were not able to fit the problem into less than 9 processors, so we decided to omit the remaining results as well. Execution times are reported for all processor

Table 4. Execution times, speedup, efficiency and MFlops rate for $n = 2520$

P	PC 10000	Origin	Exemplar
1	47.54	33.10	51.01
2	24.76	19.15	25.59
3	17.10	12.94	17.61
4	13.23	9.85	13.29
5	11.07	8.01	10.23
6	9.67	6.66	8.69
7	8.74	5.79	7.54
8	7.97	5.12	6.81
9	7.50	4.65	6.24
10	7.24	4.07	5.72
12	6.73	3.40	4.86
14	6.37	2.91	4.24
15	6.29	2.77	4.03
Speedup on 15 processors			
15	7.55	11.95	12.65
Efficiency on 15 processors			
15	0.5033	0.7966	0.8438
MFlops rate on 1 and 15 processors			
1	46.62	66.96	43.45
15	352.35	800.10	549.95

numbers between 1 and 15 that are equal divisors of the problem size.

The results are again following the general pattern established earlier. The single processor performance of all three machines becomes slightly worse suggesting that with the problem size increasing the cache memory conflicts are encountered. Similar effects are visible for the 15 processor performance, where the total MFlop rate reduces slightly on the Origin, while remains almost unchanged on the Exemplar. The efficiency of the shared memory PC 10000 is further reduced to only about 50%. This can be explained by the fact that as the problem size increases, in addition to the bus saturation, the cache memory management becomes inefficient thus further reducing the performance (see for instance [?]).

4 Concluding remarks

In this note we have reported a parallel performance comparison of a 2D elliptic solver on seven different parallel machines. Our results indicate that the shared memory (bus-based) architectures cannot deal with large problems of this type. We have also seen the effects of per-node memory size and the network performance on the overall performance of the parallel system. Finally, we have observed that the standard performance measures (speedup and efficiency) are extremely misleading when performance of multiple computer systems is compared against each other.

In the near future we plan to perform a similar set of comparisons for a 3D elliptic solver which is currently under development.

Acknowledgments

The research of the first two authors has been supported by Bulgarian NSF Grant I-701/97. Computer time grants from NCSA and NPACI are kindly acknowledged. We would like to express our gratitude to Charles A. Wright for helping us with running experiments on the NOW and the Beowulf and to C. Hempel for making the runs on the T3E possible.

References

1. G. Astfalk, HP/Convex, personal communication.
2. I. Bar-On, M. Paprzycki *Computer Assisted Mechanics and Engineering Sciences* **5**, 85 (1998).
3. R.H. Chan, T.F. Chan, *J. Numerical Lin.Alg.Appl.* **1**, 77 (1992).
4. R. Chan, G. Strang, *SIAM J.Sci.Stat.Comp.* **10**, 104 (1989).
5. J.J. Dongarra, W. Gentzsch, *Computer Benchmarks* (North-Holland, Amsterdam, 1993).
6. I. Lirkov, S. Margenov, in *Proceedings of the PARCELLA'96*, R. Vollmar, W. Erhard, V. Jossifov, eds., 279 (Akademie Verlag, Berlin, 1996).
7. I. Lirkov, S. Margenov, M. Paprzycki, R. Owens, *Large-Scale Scientific Computations of Engineering and Environmental problems*, M. Griebel, O. Iliev, S. Margenov, P. Vassilevski eds., *Notes on Numerical Fluid Mechanics*, **62** 319 (Vieweg Verlag, Braunschweig, Germany, 1998).
8. I. Lirkov, S. Margenov, M. Paprzycki, Parallel solution of 2D elliptic PDE's on Silicon Graphics supercomputers, Proceedings of the International Conference on Parallel and Distributed Computing and Systems, October, 1998, Las Vegas, Nevada, to appear.

9. I. Lirkov, S. Margenov, P.S. Vassilevski, *Computing* **53(1)**, 59 (1994).
10. I. Lirkov, S. Margenov, L. Zikatanov, *Computing* **58(3)**, 245 (1997).
11. Charles Van Loan, *Computational frameworks for Fast Fourier Transform* (SIAM, Philadelphia, 1992).