Mathematica Balkanica

New Series Vol. 13, 1999, Fasc. 1-2

Solving Structured Matrix Problems on a Cray Vector-Computer [1]

M. Paprzycki *, *C. Cyphers* **

Presented by Bl. Sendov

Block structured matrices arise from discretizations of a number of mathematical problems. Since they may need to be dealt with inside each step of an iterative process or by individual processors of a parallel computer, it is very important to solve them efficiently. A level 3 *BLAS* based library of subroutines performing basic operations (multiplication, factorisation and back substitution) is presented. The efficiency of individual subroutines is examined and compared to that of the banded solver.

AMS Subj. Classification: 68-04, 65-04, 65F05, 65Y10, 65Y20

Key Words: structured matrices, vector-computing, BLAS, LAPACK, performance

1. Introduction

Block structured matrices (block bidiagonal, block tridiagonal and almost block diagonal) arise from the discretization of a number of mathematical problems [1, 8, 14, 15, 16, 17]. There exist at least two situations when the efficiency of operations on such matrices is of extreme importance. First, in the case of non-linear problems block structured matrices may be factorized in each step of an iterative process. For instance, in the case of a Newton-type iteration the linear system solution is the most costly part of the algorithm, so any performance gain in this step will substantially reduce the total solution time [8]. Second, a number of tearing-type strategies have been designed to solve block structured linear systems on parallel computers [1, 2, 4, 10, 17]. In these methods, each processor solves a smaller problem which has the same structure as the original problem. It is thus extremely important to have an efficient solver to perform factorizations on individual processors as this is the only way to achieve overall high efficiency of parallelization.

[1] A computer time grant from the NPACI in Austin is kindly acknowledged.

In this paper, we will introduce a level 3 *BLAS* [5] based library of algorithms designed to perform basic numerical operations on three classes of block structured matrices: block tridiagonal, block bidiagonal and almost block diagonal. The library consists of a matrix-matrix multiplication routine (for normal and transposed matrices), a Gaussian elimination based decomposer (where special care has been taken to avoid fill-in) and a back solver (for the normal and transposed system). The library has a unified, *LAPACK* [3] based interface. Since each of the three matrix structures can be enclosed in a banded structure we have compared the efficiency of the new methods with the application of this technique. The performance characteristics have been studied on a Cray J-916 supercomputer. It is a small air-cooled version of the Cray Y-MP architecture. Each processor is a vector-processor capable of delivering 200 MFlops of theoretical peak performance. It was found that for matrix multiplication, when the level 3 *BLAS* routine *SGEMM* is used, a practical peak performance of 195 MFlops can be reached (see also [12]). In our experiments optimized *BLAS* and/or *LAPACK* kernels provided by the SGI Cray Computer were used. Timings were obtained using the *perftrace* utility. All results presented are averages of multiple runs.

2. Matrix multiplication

For all three structured matrices matrix multiplication is performed as a series of calls to the level 3 *BLAS* matrix multiplication routine *SGEMM* [3, 5]. For the transposed matrix the transpose option of the *SGEMM* is used. Due to the special floating point arithmetics supported by the Cray computers (single precision results in 14 digits of accuracy and is hardware supported; double precision is software emulated) single processor *BLAS* kernels have been used. However, the library contains both single and double precision routines.

The first series of experiments was performed for the matrix-vector multiplication. We have observed the effects of increasing the block size n on the performance. The results for $m = 400$ row blocks and $n = 2, 3, \ldots, 30$ are summarized in Figure 1. It should be noted that almost block diagonal matrices arise most often when boundary value ordinary differential equations (BVP ODE's) are discretized [14, 15]. It was thus assumed that finite differences have been applied to a BVP ODE leading to a system where each block is of the same size except the first and last blocks which represent boundary conditions. We experimented with two extreme cases: when boundary conditions are equally split between left and right (denoted as "n/2") and when all boundary conditions but one are on the right (denoted as "1").

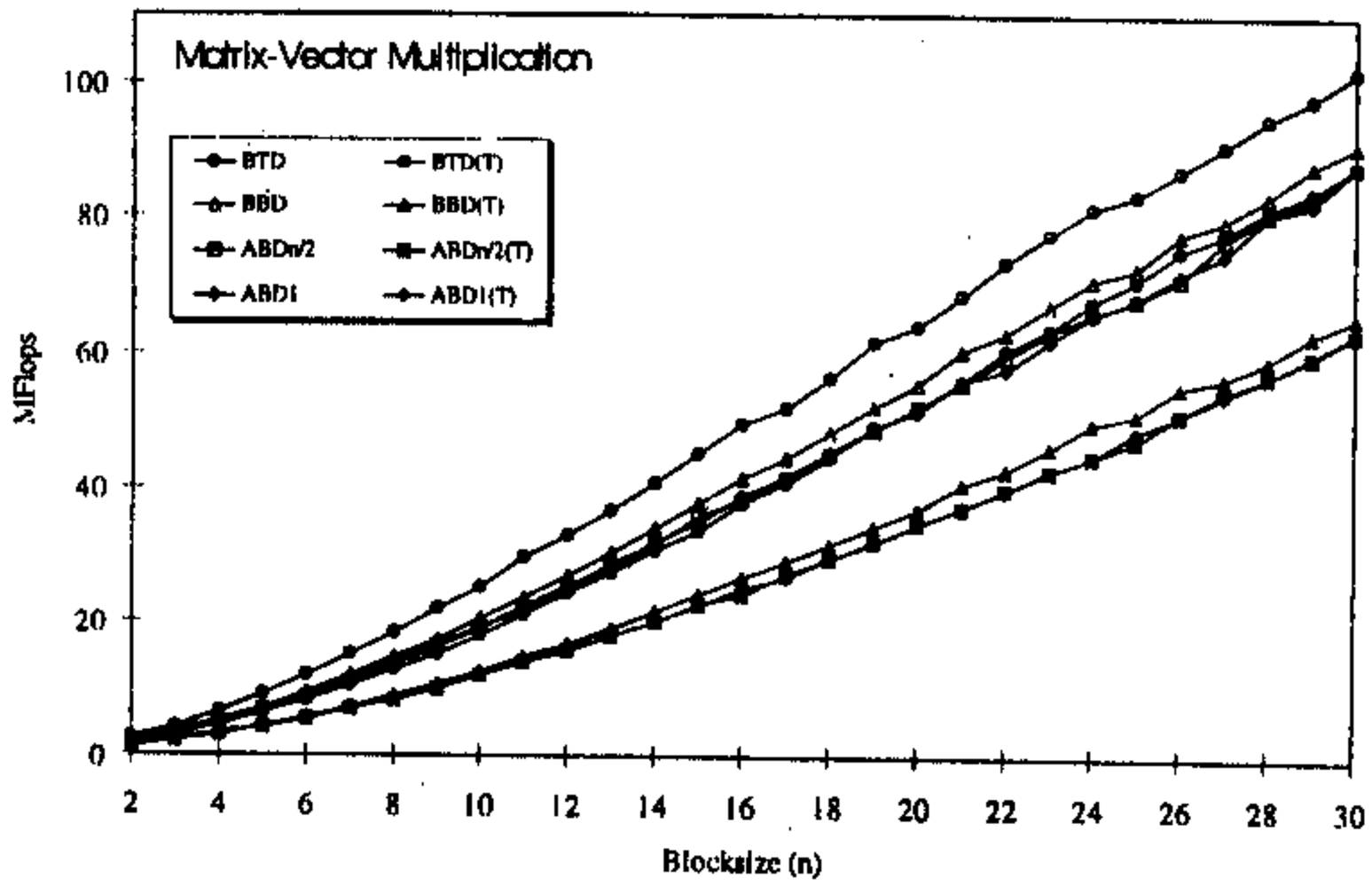

Figure 1. Matrix-vector multiplication; block tridiagonal matrices (BTD), block bidiagonal matrices (BBD) and almost block diagonal matrices (ABD); transposed multiplication denoted by (T); fixed m; results in MFlops

The results presented above, as well as in additional experiments, indicate that matrix-vector multiplication reaches approximately 120 MFlops (see also [13]). This is about 62% of the Cray's practical peak performance of 195 MFlops [12]. This should be expected, as matrix-vector multiplication is a level 2 BLAS operation and thus less efficient than level 3 BLAS operations [7, 11, 12]. The transposed multiplication is slightly less efficient because of the data access patterns. Finally, for all three block structures, the performance is independent of the number of row blocks (m).

The situation changes when the Cray-optimized level 2 *BLAS* banded matrix-vector multiplier (routine *SGBMV*) is used. Before we proceed further it should be pointed out, that the *LAPACK* banded routines use a different data storage scheme than the one proposed in the library. The library routines follow the block-orientation of the underlying matrices while the banded routines are vector-oriented, storing the main diagonal and each sub- and super-diagonal of the banded matrix as a vector of length equal to the size of the matrix. This difference effects the performance of the matrix operations. The performance of

the matrix-vector multiplication, for the matrix of bandwidth $q = 4$, increases from 40 MFlops for the matrix of order 100 to about 120 MFlops for the matrix of order 800. The performance reaches 170 MFlops for matrices of order 2800 and bandwidth $q = 18$. This makes the banded approach significantly more efficient (for small blocksizes) than the block-oriented approaches even though the banded multiplier performs a larger number of arithmetical operations. The times of the banded and block-oriented approaches become comparable only for the largest blocksizes reported above ($n = 30$). The block-oriented approaches become more efficient due to the increase of the performance related to the increase of the size of the individual blocks. In addition, the performance (compared to the banded approach) improves as no arithmetical operations are performed outside of the block structure.

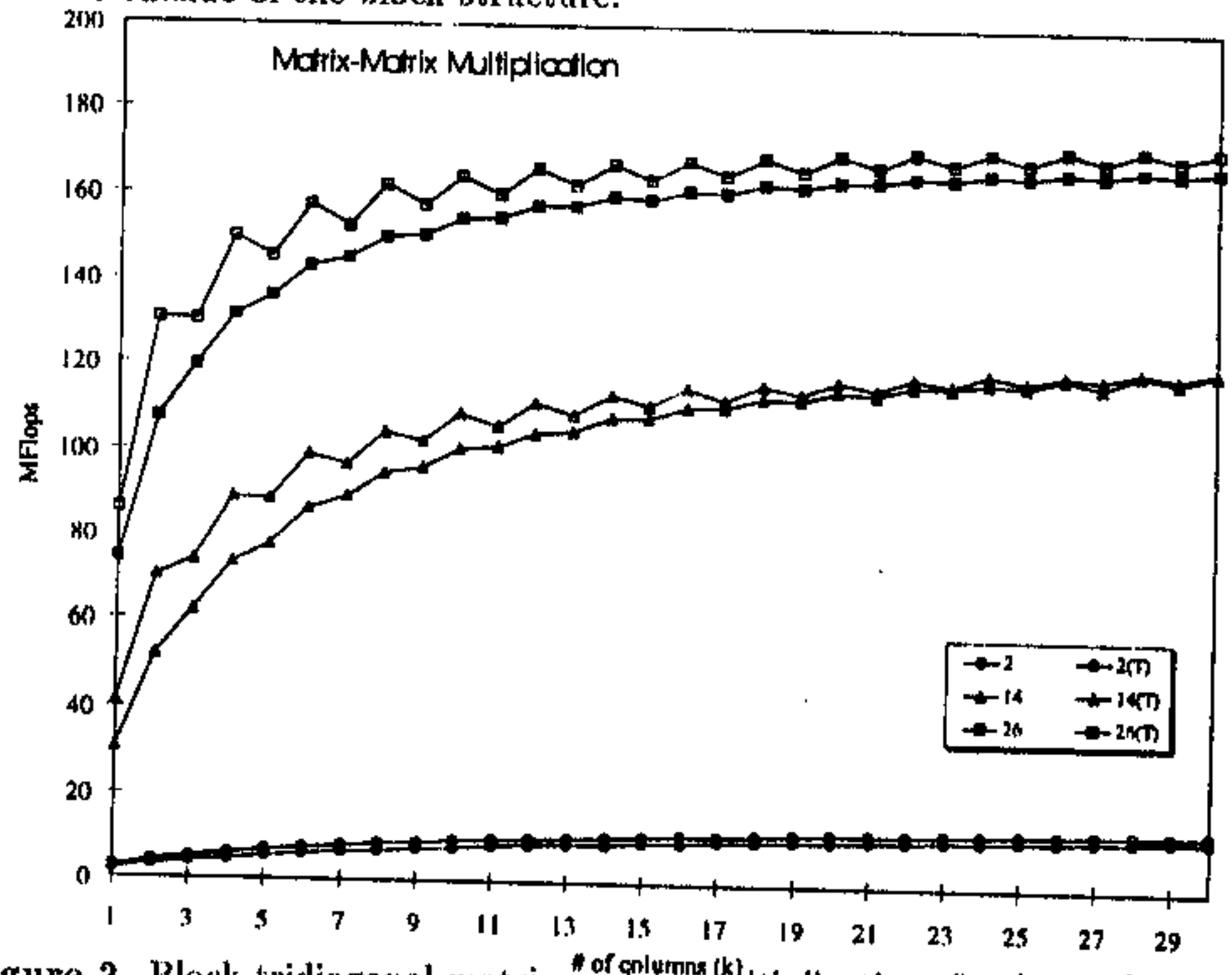

Figure 2. Block tridiagonal matrix-matrix multiplication; fixed n and m; increasing k; results in MFlops

The second series of experiments investigated the changes in the performance characteristics of the multipliers when the matrix-vector multiplication becomes matrix-matrix multiplication (we multiplied matrices $M \times B$, where M is block tridiagonal of size $mn \times mn$ while B is dense of size $mn \times k$). It should

be pointed out here that in the *LAPACK* library there is no level 3 *BLAS* banded matrix multiplication routine so no appropriate comparison could have been made. Figure 2 illustrates the effect of increasing k from 1 to 30. The results (in MFlops) are presented for $n = 2, 14, 26$ and $m = 400$. Results for the block bidiagonal and almost block diagonal matrices were quite similar and are therefore omitted.

For all three block structures the efficiency of matrix-matrix multiplication increases rapidly even for small values of k. For small n, multiplication reaches about 80% of its peak performance for $k < 25$. As n increases the value of k needed to reach the 80% of the peak performance decreases and becomes as small as 10 for $n = 26$. We have also observed that as the number of columns of B increases, the difference between the performance of normal and transposed multiplication disappears. We have experimented with matrix-matrix multiplication, where B is a full matrix and increased the block size n. The performance steadily increased from about 18 MFlops for $n = 2$, to 187 MFlops for approximately $n = 40$ where it stabilized (reaching 96% of the practical peak performance). Finally, as was the case for matrix-vector multiplication, that the number of blocks (m) has almost no effect on the performance.

3. Matrix factorization

We consider the solution of a linear system $Mx = b$, where the matrix M is block structured. The matrix M is factorized using a blocked version Gaussian elimination which has been modified to match the particular block structure to avoid generating fill-in. The algorithms consist of a series of calls to the *LAPACK* routine *SGETRF* (performing LU decomposition), and level 3 *BLAS* routines *STRSM* (performing back substitution for multiple right hand sides) and *SGEMM* (performing matrix-matrix multiplication). The details of such block decompositions can be found in [1, 6, 13, 14, 15, 16]. The stability analysis of these algorithms was developed by Varah [15, 16]. In general, his results indicate that in the case of almost block diagonal systems the block algorithm is as stable as Gaussian elimination with partial pivoting. For the block bidiagonal and block tridiagonal matrices the decomposition is stable if the matrices are block diagonally dominant.

Figure 3 presents the performance of the factorization routines for all three matrix structures. In the case of block bidiagonal and block tridiagonal matrices the performance of decomposers with row and column pivoting is presented (indicating the column-pivoting version with "(c)"). As above, for the almost block diagonal systems two cases corresponding to two extreme distributions of boundary conditions are presented. The results for $m = 400$ block

rows and for the blocksize n increasing from 2 to 30 are reported in MFlops.

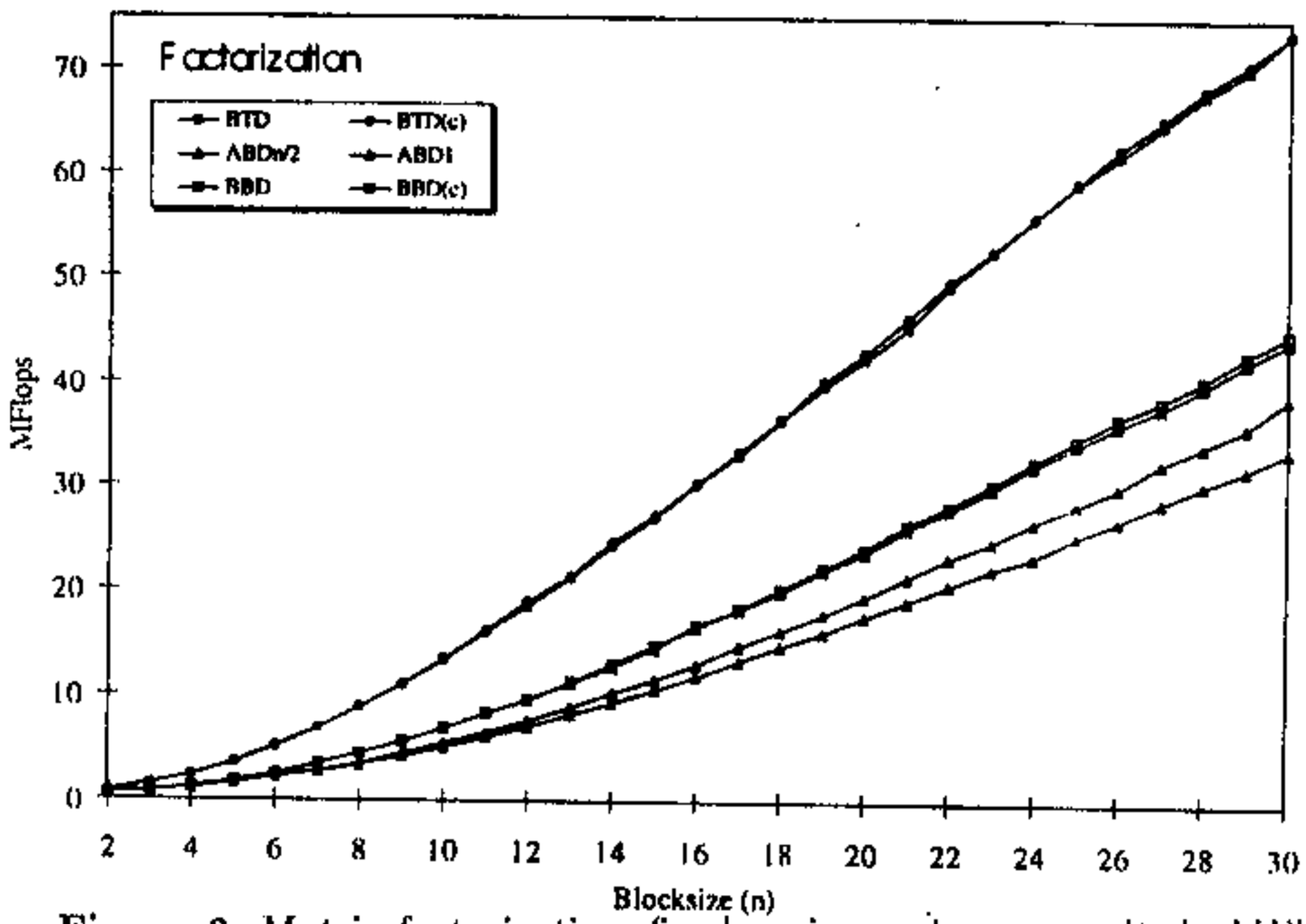

Figure 3. Matrix factorization; fixed m; increasing n; results in MFlops.

The factorization routines are much less efficient than matrix multiplication. As the block size increases the efficiency of factorization improves from 0.8 MFlops for $n = 2$ to about 30 MFlops for almost block diagonal matrices and to approximately 70 MFlops for block tridiagonal matrices for $n = 30$. This can be explained by the fact that the almost block diagonal decomposer is effectively operating on blocks half the size of the blocks in the remaining algorithms (see [6, 13, 14] for more details). Both the row- and column-oriented versions behave similarly while the effect of unbalanced boundary conditions are clearly visible for the almost block diagonal matrices. Our experiments indicate that, as before, the number of block rows m has almost no effect on performance.

Due to the objectivity considerations the comparison with the banded approach is slightly more difficult. The *LAPACK* library contains the routine *SGBTRF* which computes an LU factorization of a banded matrix. This routine uses pivoting and row interchanges to assure stability. This means that fill-in (of the size of the upper bandwidth) is generated and additional arithmetic operations are performed. Since pivoting is performed also by the block-oriented algorithms it seems to be fair to make a straightforward comparison with the block oriented algorithms. It is assumed that the additional operations is the

price that one has to pay for using the library "black-box" software. In Table 1 we compare the time it takes for the block tridiagonal and banded approaches to factorize a matrix of size mn where $m = 400$ and $n = 2, 6, 10, 14, 18$. The results for the remaining two block structures are similar and are thus omitted.

n	2	6	10	14	18
BAND	0.0168	0.071	0.157	0.274	0.440
BTD	0.0521	0.096	0.156	0.226	0.316
Ratio	0.3225	0.738	1.006	1.212	1.392

Table 1. Matrix factorization; comparing block tridiagonal and banded approaches; $m = 400$; time in seconds.

It can be observed that for narrow matrices the banded approach outperforms the block oriented algorithm but as the bandwidth increases the situation reverses. For the largest blocksize reported here ($n = 30$) the block oriented approach becomes twice faster than the banded routine.

4. Back substitution

The back substitution consists of a number of calls to the level 3 *BLAS* routine *STRSM*. For the block tridiagonal and block bidiagonal matrices there are four versions of back solvers corresponding to standard and transposed solution for the row and column pivoting strategies. Since no performance difference was observed between the row and column oriented versions only the row oriented results will be reported. The first series of experiments studied the performance of back solvers for $m = 400$, one right hand side, and increasing block size $n = 2, 3, ..., 30$. The results (in MFlops) are presented in Figure 4.

As the block size increases the performance of the back solvers improves from 1.2 MFlops for $n = 2$ to between 8 and 14 MFlops for $n = 30$. The performance is relatively low for two reasons. First, the sizes of individual blocks are relatively small. Second, for one right hand side the performance the level 3 *BLAS* routines is effectively reduced to that of level 2 *BLAS* kernels. All back solvers behave similarly independent of their particular block structure. The comparison with the banded approach (*LAPACK* routine *SGBTRS*) is, for the reasons stated above, relatively complicated. It has been observed that while the straightforward backsubstitution is substantially faster than its blocked counterpart, the transposed backsubstitution runs at about the same speed as the

block-oriented routines. This situation changes when multiple right hand sides are considered. Here, even though the banded solver *SGBTRS* allows for the solution for the multiple right hand sides, no performance gain from moving from level 2 to level 3 *BLAS* has been observed. This is not the case for the block-oriented algorithms. Figure 5 summarizes the performance results for the back solvers for the block bidiagonal matrices (similar behavior has been observed for the remaining block structures) for $n = 2, 14, 26$, $m = 100$ and an increasing number of right hand sides $RHS = 1, 2, \ldots, 30$.

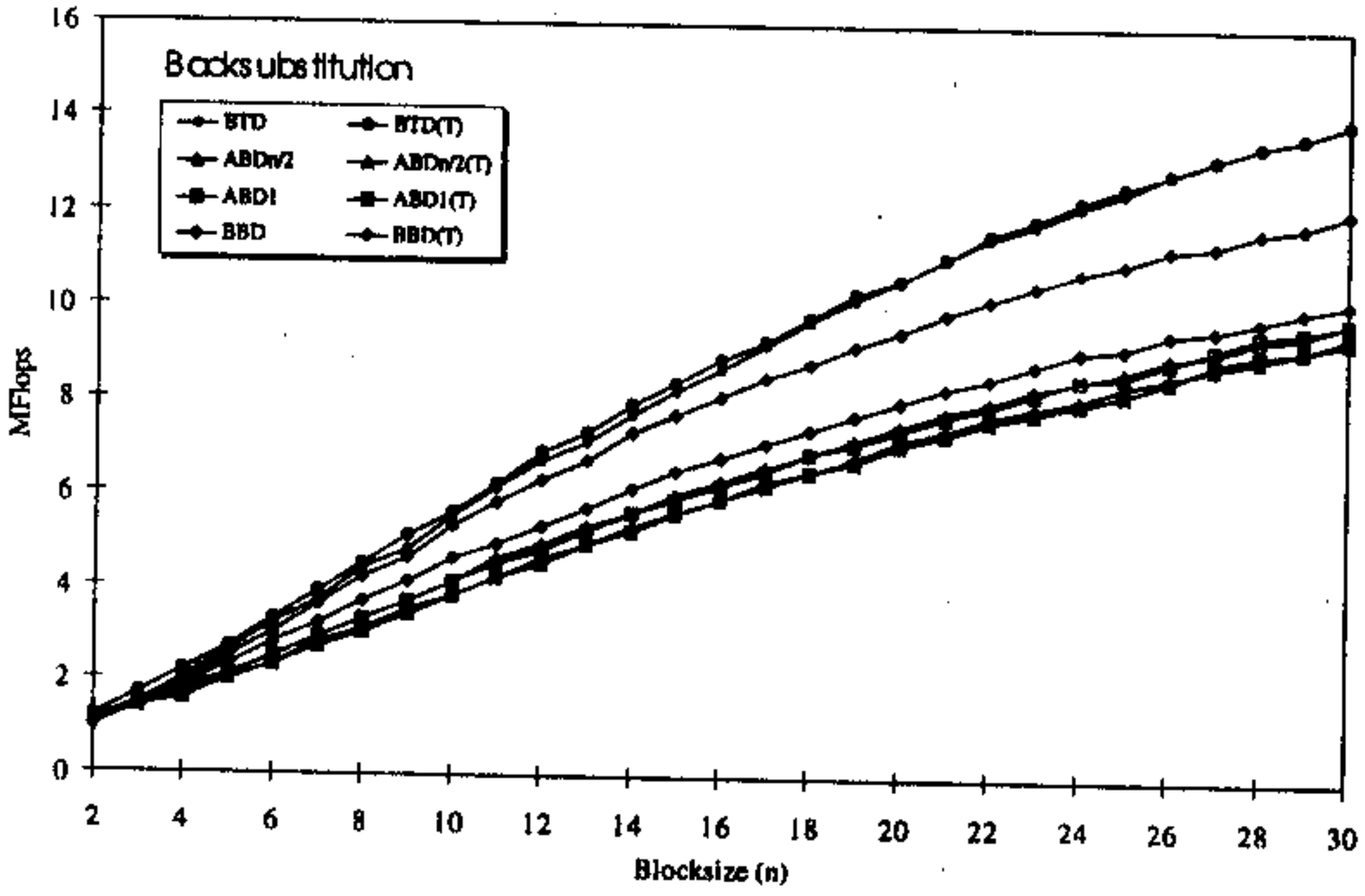

Figure 4. Back substitution; one right hand side; fixed m; increasing n; results in MFlops.

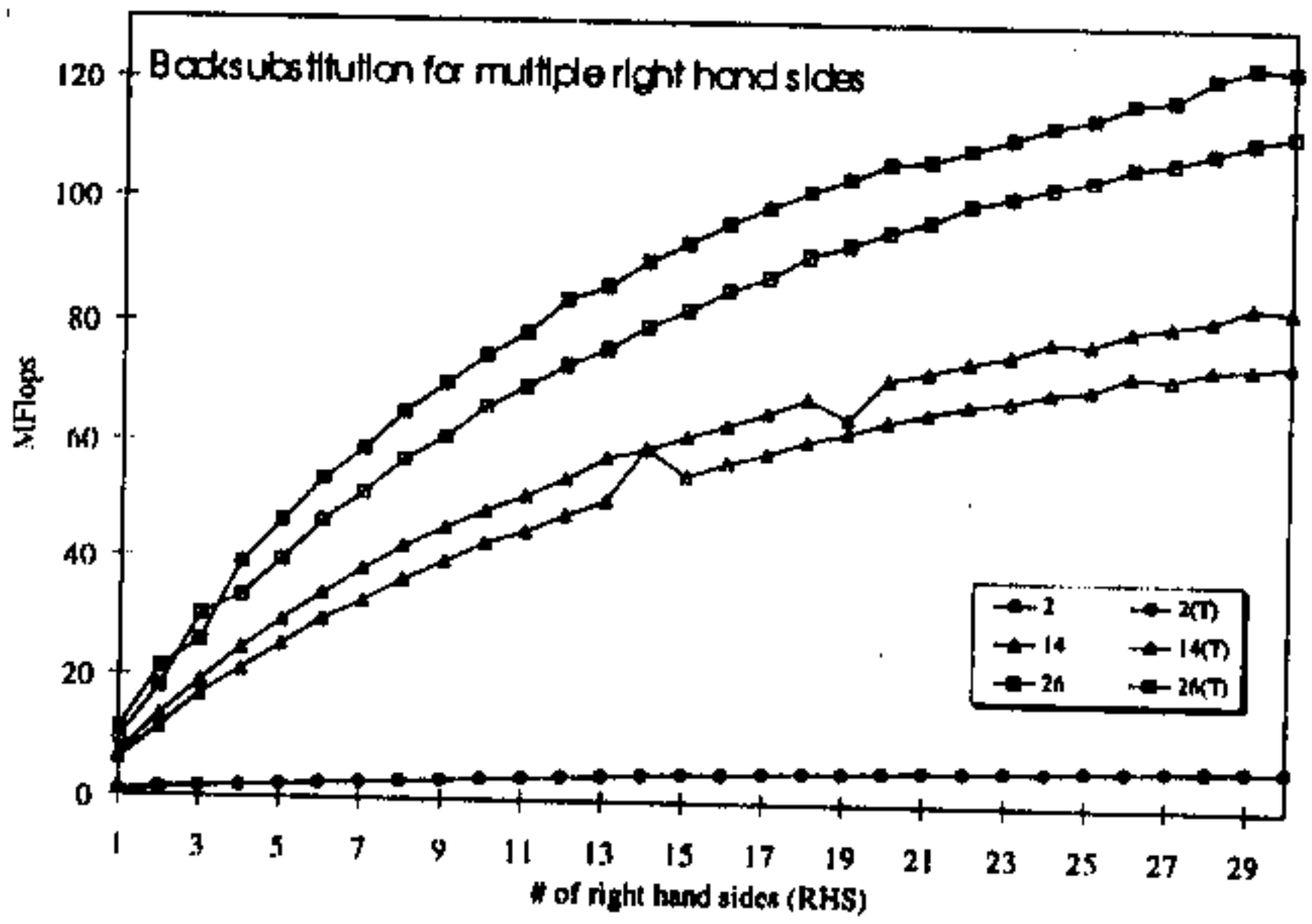

Figure 5. Back substitution; fixed m; increasing the number of right hand sides RHS; results in MFlops.

The results are somewhat similar to these presented on Figure 2; however, the performance gain from moving from level 2 to level 3 $BLAS$ is less immediate for small values of RHS. Surprisingly, the transposed version of the solver is more efficient. This can be attributed to the imbalance in the structure of the block bidiagonal matrix. For the block tridiagonal and almost block diagonal (for the balanced boundary conditions) solvers the performance in cases of standard and transposed backsubstitution was almost identical. Since the banded solver has not provided any performance gains for multiple right hand sides, it has been outperformed by the block-oriented solvers.

5. Conclusion

A library of subprograms for performing operations on block structured matrices has been presented and its performance compared to the usage of black-box $BLAS/LAPACK$ based routines. The results of comparisons are mixed. For very narrow block structures the application of highly optimized

vector-oriented banded routines seems to be the best solution. As the blocksizes increase the block oriented algorithms outperform the banded approach. This is especially important when, for instance, the spectral methods are applied to the solution of partial differential equations on rectangular domains [8, 9]. Here the linear systems consist of a moderate number of large blocks and the information about the matrix structure has to be utilized to achieve an efficient solution to the problem.

In the next step we plan to experiment with the RISC-based supercomputers and see how much of the performance characteristics is related to the specific optimizations provided in Cray's scientific computing library.

References

[1] P. Amodio, T. Politi, M. Paprzycki. Survey of parallel algorithms for block bidiagonal linear systems. *J. of Computers in Mathematics Applications*, **31**, No 7, 1996, 111-127.

[2] P. Amodio, T. Politi, M. Paprzycki. Solving block bidiagonal linear systems on distributed memory computers. *Proc. of Seventh Internat. ISCA Conference on Parallel and Distributed Computing Systems*, ISCA Press, Raleigh, NC, 1994, 812-815.

[3] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. DuCroz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov and D. Sorensen. *LAPACK Users' Guide*, SIAM, Philadelphia, 1993.

[4] M. Berry and A. Sameh. Multiprocessor schemes for solving block tridiagonal linear systems. *The International J. of Supercomp. Appli.*, **2**, 1988, 37-57.

[5] J. Dongarra, J. DuCroz and S. Hammarling. *A Set of Level 3 BLAS Basic Linear Algebra Subprograms*. Technical Report ANL-MCS-TM57, Argonne National Laboratory, 1988.

[6] C. Cyphers, M. Paprzycki and I. Gladwell. *A Level 3 BLAS Based Solver for Almost Block Diagonal Linear Systems*. Software Report 92-3, Southern Methodist University, 1992.

[7] C. Cyphers and M. Paprzycki. Gaussian elimination on a Cray Y-MP. *CHPC Newsletter*, **6**, 1991, 43-47.

[8] C. Cyphers, M. Paprzycki, A. Karageorghis. High performance solution of partial differential equations discretized using a Chebyshev spectral collocation method. *J. of Computational and Applied Mathematics*, **66**, No 1, 1996, 71-80.

[9] A. K a r a g e o r g h i s, M. P a p r z y c k i. An efficient direct method for fully conforming spectral collocation schemes. *Numerical Algorithms*, **12**, 1996, 309-319.

[10] V. M e h r m a n n. Divide and conquer methods for block tridiagonal systems. *Parallel Computing*, **19**, 1993, 257-279.

[11] M. P a p r z y c k i. Comparison of Gaussian elimination algorithms on a Cray Y-MP, *Linear Algebra and Applications*, **172**, 1992, 57-69.

[12] M. P a p r z y c k i and C. C y p h e r s. Multiplying matrices on the Cray – Practical considerations. *CHPC Newsletter*, **6**, 1991, 77-82.

[13] M. P a p r z y c k i and C. C y p h e r s. Level 3 BLAS based library for block tridiagonal matrices. *Proc. of 2nd Conference on Difference Equations with Applications*, Vesprem, Hungary, 1995.

[14] M. P a p r z y c k i, I. G l a d w e l l. Solving almost block diagonal systems using level 3 BLAS. *Proc. of Fifth SIAM Conference on Parallel Processing for Scientific Computing*, SIAM, Philadelphia, 1992, 52-62.

[15] J. V a r a h, Alternate row and Column Elimination for Solving Certain Linear Systems, *SIAM J. Numer. Analysis*, **13**, 1976, 71-75.

[16] J. V a r a h, On the solution of block-tridiagonal systems arising from certain finite-difference equations. *Mathematics of Computation*, **26**, 1972, 859-868.

[17] K. W r i g h t, Parallel treatment of block-bidiagonal matrices in the solution of ordinary differential boundary value problems, *J. of Computational and Applied Mathematics*, **45**, 1993, 191-200.

Received 19.12.1997

* *Dept. of Computer Science and Statistics*
University of Southern Mississippi
Hattiesburg, MS 39406-5106, USA
e-mail: marcin.paprzycki@usm.edu

** *Teraco Inc.*
Midland, TX 79760, USA
e-mail: cyphers@acm.org