HIGH PERFORMANCE SOLUTION TO STRUCTURED LINEAR SYSTEMS

MARCIN PAPRZYCKI
CLIFF CYPHERS
Department of Mathematics and Computer Science
University of Texas of the Permian Basin
Odessa, TX 79762
USA

ABSTRACT
Structured linear systems arise from discretizations of a number of mathematical problems. Since they can be solved in each step of an iterative process it is very important to solve them efficiently. Two level 3 BLAS based algorithms for the solution of structured linear systems (block tridiagonal and almost block diagonal) are presented. Their efficiency is studied.

KEYWORDS: Structured linear systems, direct solution, BLAS, software library, efficiency

1. INTRODUCTION

Structured linear systems arise from the discretization of a number of mathematical problems [1,2,3]. A structured linear system is a large sparse system assembled out of a number of rectangular and possibly triangular blocks. It is assumed that each of these blocks is dense. Examples of such systems include banded, block bidiagonal, block tridiagonal, almost block diagonal systems and others. In case of non-linear problems such systems are solved in each step of the iterative process. Since these systems are large and sparse a number of tearing-type strategies have been designed to solve them on parallel computers [4,5,6]. Typically, each processor solves a smaller problem of 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 the overall high efficiency of parallelization. In the case of a Newton-type iteration the linear system solution is the most costly part of the process, so any performance gain in this step will substantially reduce the total solution time.

In this paper, we will introduce two level 3 BLAS [7] based families of algorithms designed to perform basic operations on block tridiagonal and almost block diagonal linear matrices. Each family consists of a matrix-matrix multiplication routine (for the basic and transposed matrices), a Gaussian elimination based decomposer (where special care has been taken to eliminate the fill-in) and a back solver (for the normal and transposed

system). Each family of algorithms will have the form of a library with a unified, LAPACK [8] based interface. The performance of the proposed libraries will be illustrated on a Cray Y-MP supercomputer.

2. BLOCK TRIDIAGONAL MATRIX LIBRARY

2.1. Matrix Factorization

We consider the solution of a linear system $M\underline{x} = \underline{b}$, where matrix M is block tridiagonal of order $N = nm$, where n denotes the number of blocks and m denotes the size of each square block:

$$M = \begin{pmatrix} A_1 & C_1 & & & & & \\ B_2 & A_2 & C_2 & & & & \\ & B_3 & A_3 & C_3 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & B_{n-1} & A_{n-1} & C_{n-1} \\ & & & & & B_n & A_n \end{pmatrix}$$

Matrix M can be factorized using block-Gaussian elimination. Each factorization step i (except the last one) involves a 4-block submatrix of M

$$\begin{pmatrix} A_i & C_i \\ B_{i+1} & A_{i+1} \end{pmatrix}.$$

In the first step of the factorization, block A_i will be $L_i U_i$ decomposed using Gaussian elimination with partial pivoting. To avoid the generation of fill-in, pivoting is performed only inside this block. The decomposition is performed using a call to the LAPACK routine _GETRF [8]. To optimize the performance of the algorithm, the decomposition is performed in a blocked fashion if m (the size of the block A_i) is larger than the optimal block size for the given machine or in an unblocked fashion otherwise. In the second step, block B_{i+1} is multiplied from the left by U_i^{-1} (resulting in B^*_{i+1}) and block C_i is multiplied from the right by L_i^{-1} (resulting in C_i^*). These two steps are accomplished by back substitution using calls to the level 3 BLAS routine _TRSM [7]. In the final step, block A_{i+1} is updated by $B^*_{i+1} C_i^*$ by calling the level 3 BLAS routine _GEMM [7]. The steps of the algorithm are then repeated starting from the just updated A_{i+1}. In the case of the last block (A_n) only the factorization step is applied. We have developed two versions of this algorithm with row pivoting and with column pivoting.

It is clear that the algorithm can be unstable as the pivoting is applied only inside the diagonal blocks. The following result by Varah [2] provides the stability condition for the proposed algorithm. It can be shown that if matrix M is block diagonally dominant the block decomposition algorithm described above is numerically stable [2], where matrix M is *block diagonally dominant* with respect to the matrix norm $\|\cdot\|$ if: $\|A_i^{-1}\|(\|B_i\| + \|C_i\|) \leq 1$, where $i = 1, \ldots, n$.

In addition to the decomposition routine described above we have developed a matrix-matrix multiplication routine that multiplies the block tridiagonal matrix (or its transpose) by another matrix (or vector). (Such a routine can be used in an iterative solver, as well as in an iterative refinement step.) We have also implemented two versions of a back solver: for the solution of the original linear system and for the solution of its transpose. All these routines have a unified interface based on the interfaces used in the LAPACK project.

2.2. Numerical Experiments

We have experimented with the library of routines for the tridiagonal matrices on a Cray Y-MP 8/864 supercomputer. The Cray provided optimized BLAS kernels were used. Timings were obtained using the *perftrace* utility. All results presented are averages of multiple runs. Figure 1 presents the results of increasing the size of these blocks $m = 79, \ldots, 129$ and multiplying a block tridiagonal matrix and its transpose by a full matrix for $n = 19$ blocks. The matrix multiplication routine is very efficient. Since the practical peak performance of a one-processor Cray Y-MP is approximately 315 MFlops [9], the matrix multiplication routine reaches 97% of this peak. For the multiplication of the transposed matrix (which utilizes calls to the transposed version of the level 3 BLAS routine _GEMM) the effects of the memory section conflicts can be observed. This can be explained by the fact that in the case of a transposed matrix multiplication the blocks of M are accessed in the row-major order.

Figure 2 presents the performance factorization routines (with row and column pivoting) for $n = 19$ blocks and increasing block sizes $m = 79, \ldots, 129$. The factorization routines are not as efficient as the matrix multiplication routines. Their efficiency increases as the block size increases and reaches approximately 85% for $m = 127$. Both the row- and column-oriented versions behave similarly with a slight advantage of the row-oriented one. Typical effects of memory section conflicts can be observed in both routines for $m = 80, 96,$

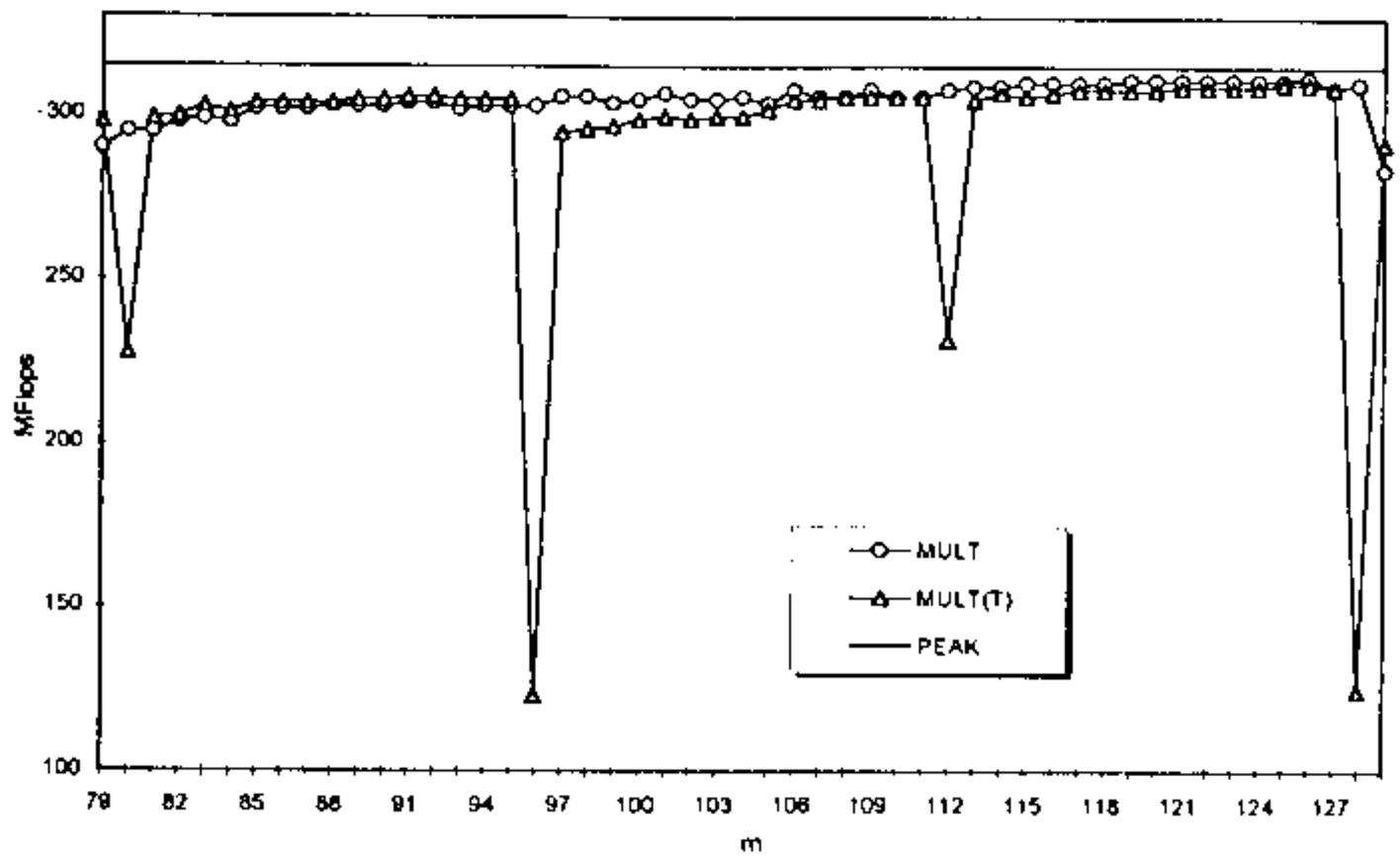

Figure 1. Matrix-matrix multiplication; n = 19; results in Mflops.

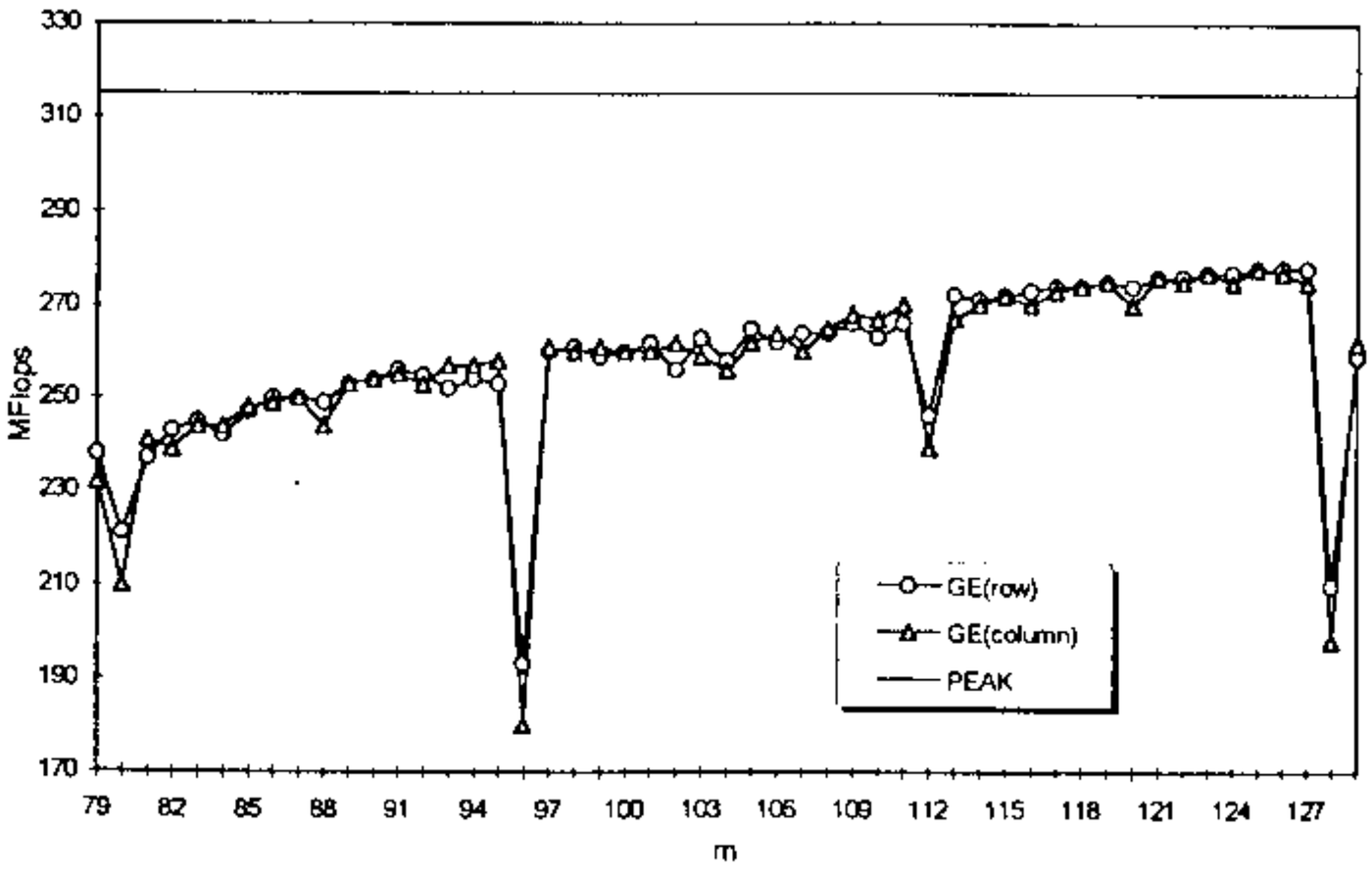

Figure 2. Performance of the Gaussian elimination routines; n = 19; results in MFlops.

112 and 128 (the increase of the block size by 16) with the performance dips being especially visible at m = 96 and 128 (the increase of the block size by 32). The results are in agreement with [10, 11].

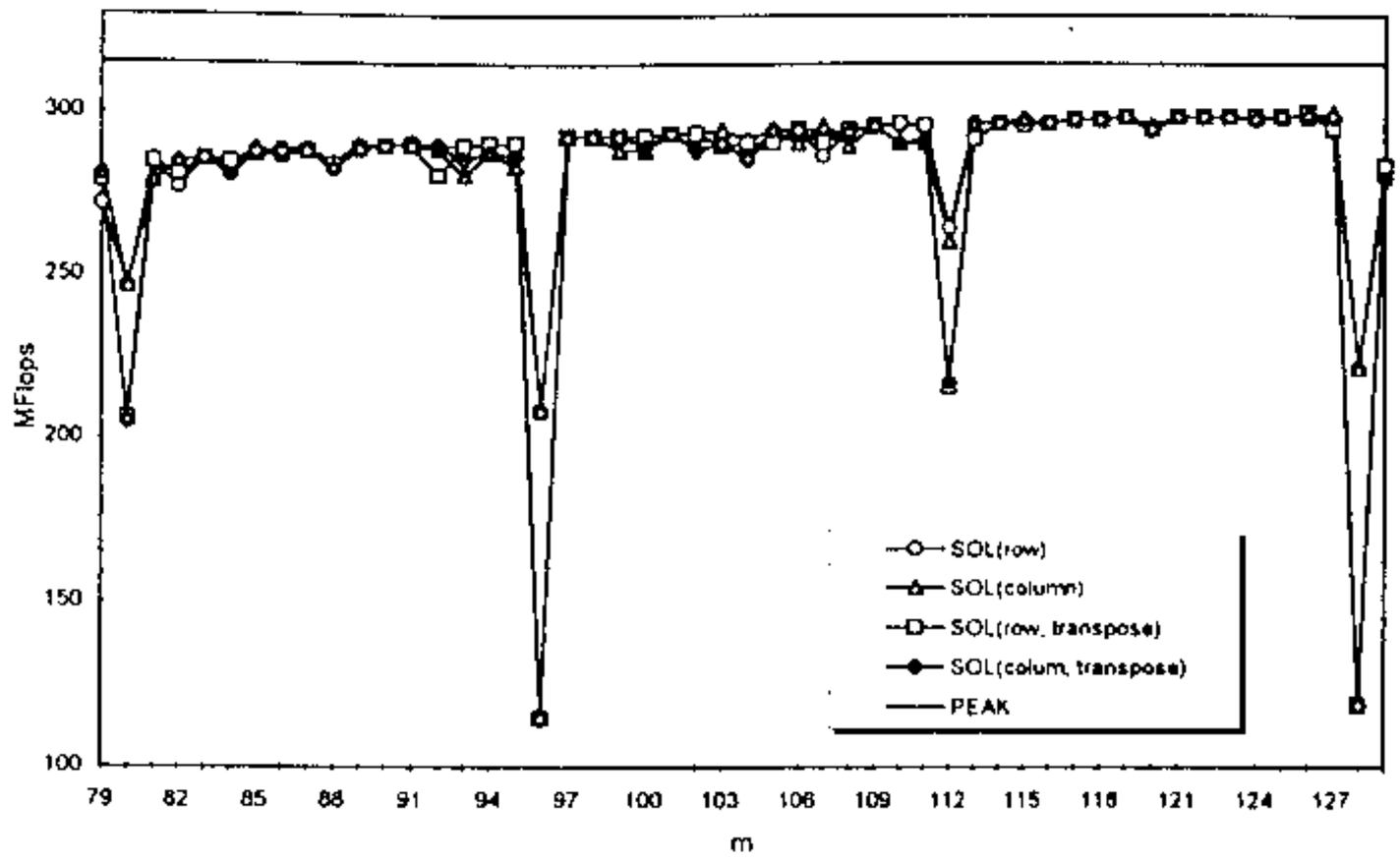

Figure 3. Performance of back solvers; n = 19 blocks, RHS = 50; results in MFlops.

Figure 3 illustrates the performance of the back solvers for n = 19 blocks, increasing block sizes m = 79, ..., 129 and for RHS = 50 right hand sides. The performance reaches almost 300 MFlops (approximately 95% of the practical peak) and increases slightly as the size of the blocks increases. The memory section conflicts are much more pronounced for the transposed versions of the back solver for the same reason as in the case of the transposed matrix multiplication.

In the next series of experiments we have studied the effects of changing the system size n and the number of right hand sides RHS on performance. In Table 1 the effects of changes of the system size (increase in number of blocks) n = 4, ..., 9 for m = 100 and RHS = 50 are presented. As could be expected increasing the number of blocks has almost no effect on the performance of the library routines. The only performance effect that can be observed is for n = 8 where the system size becomes divisible by 32 and thus the effects of memory section conflicts can be observed.

Table 1. Effects of changes in the system size, results in MFlops.

n =	4	5	6	7	8	9
MULT	306	306	306	307	306	307
MULT(T)	298	299	299	299	299	299
GE(col)	258	261	261	264	263	265
GE(row)	256	259	260	262	261	263
SOL(col)	292	262	293	294	214	294
SOL(row)	292	263	293	294	214	294
SOL(col, T)	292	262	293	294	213	294
SOL(row, T)	292	262	293	294	214	294

Table 2 illustrates the effects of increasing the number of columns of the matrix that the block tridiagonal matrix *M* is multiplied by (from 1 — representing a matrix-vector multiplication, through 25 to 50 columns) as well as of increasing the number of right hand sides (1, 25, 50) on the performance of the back solvers The results were collected for *n* = 8 blocks of size *m* = 100. There is a clear performance increase as the number of

Table 2. Effects of change in the number of columns routines operate on; results in MFlops

# of columns (RHS)	1	25	50
MULT	288	306	306
MULT(T)	266	298	299
SOL(row)	134	206	214
SOL(col)	133	206	214
SOL(row, T)	133	206	214
SOL(col, T)	133	206	214

columns increases. The results for matrix-vector multiplication and for the solution of the subsystem with one right hand side illustrate the efficiency decrease observed when moving from a blocked level 3 BLAS based algorithm to an unblocked level 2 BLAS based algorithm. These results are in agreement with [10, 11].

3. ALMOST BLOCK DIAGONAL MATRIX LIBRARY

We assume that matrix *M* is almost block diagonal (ABD) as defined in [3, 12] The proposed algorithm is a level 3 BLAS based extension of that proposed in [1, 13]. We will describe it only briefly as a more detailed description can be found in [3, 12]. For our purposes we will rewrite the ABD system as

$$M = \begin{pmatrix} A_{11} & A_{12} & & & & & & \\ A_{21} & A_{22} & A_{23} & A_{24} & & & & \\ A_{31} & A_{32} & A_{33} & A_{34} & & & & \\ & & \ddots & \ddots & \ddots & & & \\ & & & \ddots & \ddots & \ddots & & \\ & & & & A_{L-3,L-3} & A_{L-3,L-2} & A_{L-3,L-1} & A_{L-3,L} \\ & & & & A_{L-2,L-3} & A_{L-2,L-2} & A_{L-2,L-1} & A_{L-2,L} \\ & & & & & & A_{L-1,L-1} & A_{L-1,L} \end{pmatrix}$$

where $A_{i,j}$ are square and $A_{i,j}$ are rectangular blocks of varying sizes. This system has thus a total of $n = L/2$ blocks. The i-th step of the algorithm consists of two phases. In phase I the rectangular block

$$\begin{pmatrix} A_{2i-1,2i-1} \\ A_{2i,2i-1} \end{pmatrix}$$

is decomposed using Gaussian elimination with partial pivoting and row interchanges into

$$P \begin{pmatrix} L_{2i-1,2i-1} \\ L_{2i,2i-1} \end{pmatrix} U_{2i-1,2i-1},$$

where P is the permutation matrix. After this factorization, block

$$\begin{pmatrix} A_{2i-1,2i} & A_{2i-1,2i+1} \\ A_{2i,2i} & A_{2i,2i+1} \end{pmatrix}$$

will be updated by the inverse of

$$\begin{pmatrix} L_{2i-1,2i-1} & 0 \\ L_{2i,2i-1} & I \end{pmatrix}.$$

In phase II, the block

$$\begin{pmatrix} A_{2i,2i} & A_{2i,2i+1} \end{pmatrix}$$

will be decomposed using Gaussian elimination with partial pivoting and column interchanges into

$$L_{2i,2i} \begin{pmatrix} A_{2i,2i} & A_{2i,2i+1} \end{pmatrix} Q,$$

where Q is the permutation matrix. After this factorization, block

$$\begin{pmatrix} A_{2i+1,2i} & A_{2i+1,2i+1} \\ A_{2i+2,2i} & A_{2i+2,2i+1} \end{pmatrix}$$

will be updated by the inverse of block

$$\begin{pmatrix} U_{2i,2i} & U_{2i,2i+1} \\ 0 & I \end{pmatrix}.$$

The factorization will be performed in a block fashion using LAPACK [1] provided routine _GETRF (see the comments in Section 2.1 above). The update steps consist of calls to the

level 3 BLAS routines _TRSM and _GEMM. As was shown in [1] the stability of the proposed algorithm will be the same as the stability of any Gaussian elimination with partial pivoting. As in the case of block tridiagonal systems, the proposed family of matrix manipulation routines consists of the matrix-matrix multiplication routine (for the basic and transposed matrices), the factorization routine described above, and back solvers for the basic and transposed system. All these routines have a unified, LAPACK based, interface.

3.2. Numerical Experiments

Numerical experiments have been performed in the same arrangements as described in Section 2.2 above. Without loss of generality it was assumed that the ABD

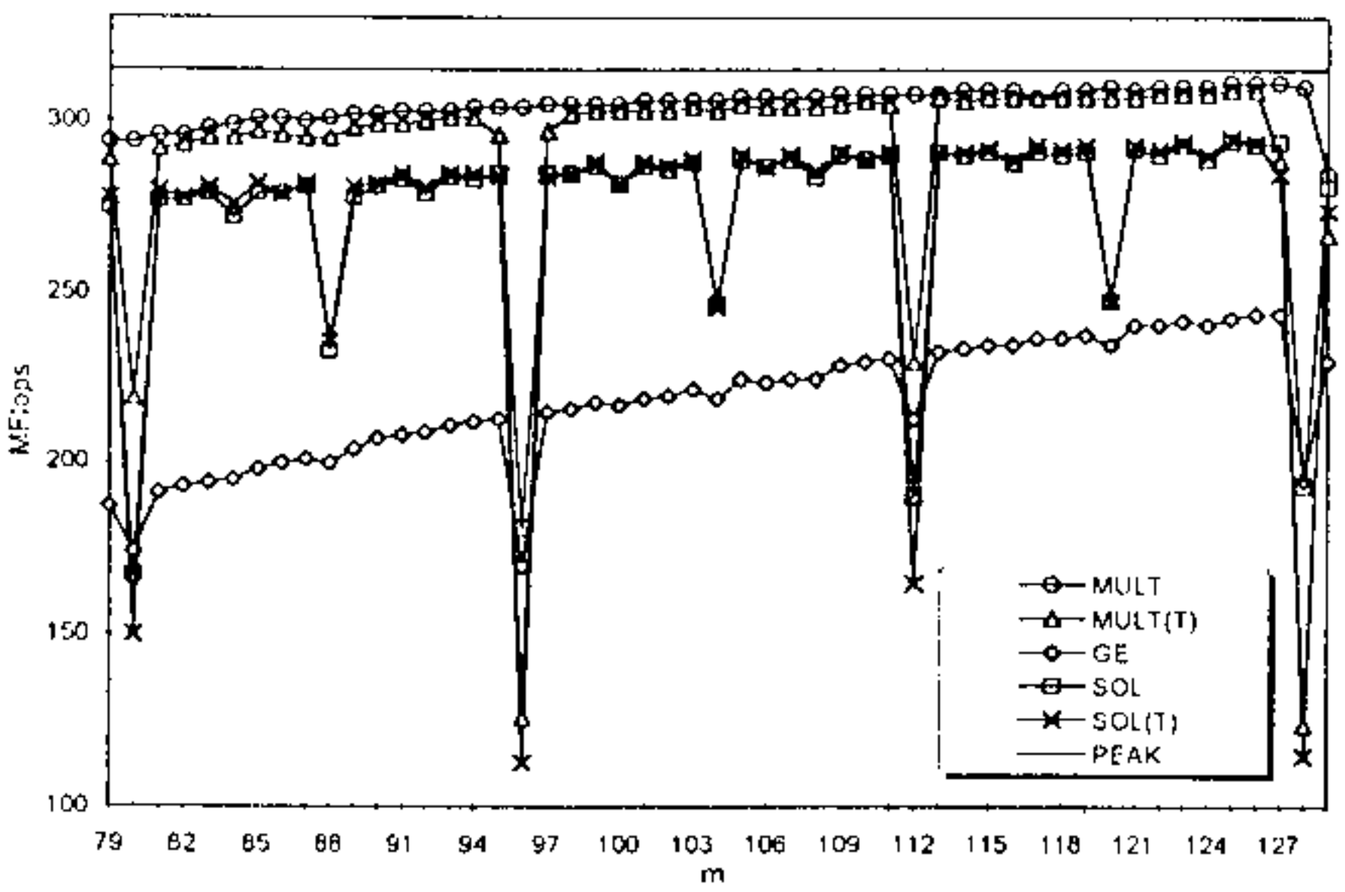

Figure 4. Almost Block Diagonal matrix library performance; $n = 7$; $RHS = 50$; results in MFlops.

system results from a finite difference discretization of a system of m first order differential equations with separated boundary conditions. The first and the last block thus have size $m/2 \times m$ while all the remaining blocks have sizes $m \times 2m$. The results are summarized in Figure 4 for $n = 7$ blocks, $RHS = 50$ right hand sides and increasing block sizes $m = 79, \ldots, 129$ The results are quite similar to those for the block tridiagonal linear systems. In this case, however, the performance of the Gaussian elimination routine reaches only

approximately 82% of the practical peak performance. This decrease can be attributed to the fact that due to the matrix structure the block operations are performed on smaller blocks (see Section 3.1 above). The performance of the matrix multiplication routine reaches 97% and the performance of the back solver for 50 right hand sides 94% of the practical peak. In a separate experiment we have established that the performance of the back solver for one right hand side reaches approximately 75% of the peak.

4. CONCLUSION

Two subroutine families for performing operations on structured linear systems have been presented. For large enough blocks they reach more than 75% efficiency. This makes them good candidates for the usage on one processor systems as well as parts of the parallel tearing-type algorithms. We were also able to observe the effects of memory section conflicts typical for the Cray Y-MP architecture. The performance involving transposed systems is severely impaired by the row-oriented matrix operations. These effects can be avoided by choosing odd leading dimension of the matrices involved.

ACKNOWLEDGMENTS

A computer time grant from the Center for High Performance Computing in Austin is kindly acknowledged.

REFERENCES

1. J. Varah, SIAM J. Numer. Analysis, 13, pp. 71-75 (1976).

2. J. Varah, Mathematics of Computation, 26, pp. 859-868 (1972).

3. M. Paprzycki and I. Gladwell, Proceedings of The Fifth SIAM Conference on Parallel Processing for Scientific Computing, SIAM, Philadelphia, (1992) pp. 52-62.

4. M. Paprzycki and I. Gladwell, Parallel Computing, 17, pp. 133-153 (1991).

5. V. Mehrmann, Parallel Computing, 19, pp. 257-279 (1993).

6. M. Berry and A. Sameh, The International J. of Supercomp. Appli., 2, pp. 37-57 (1988).

7. J. Dongarra, J. Du Croz and S. Hammarling, Technical Report ANL-MCS-TM57, Argonne National Laboratory (1988).

8. E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov and D. Sorensen, LAPACK Users' Guide, SIAM, Philadelphia (1993).

9. M. Paprzycki and C. Cyphers, CHPC Newsletter, 6, pp. 77-82 (1991).

10. C. Cyphers and M. Paprzycki, CHPC Newsletter, 6, pp. 43-47 (1991).

11. M. Paprzycki, Lin. Alg. and Applications, 172 pp. 57-69 (1992).

12. C. Cyphers, M. Paprzycki and I. Gladwell, Software Report 92-3, Southern Methodist University (1992).

13. J. Diaz, G. Fairweather and P. Keast, ACM Trans. Math. Software, 9, pp. 359-375 (1983).