Proceedings of the ISCA
10th International Conference

COMPUTER APPLICATIONS IN INDUSTRY AND ENGINEERING

San Antonio, Texas U.S.A.
December 10 - 12, 1997

Editor: T. Philip

A Publication of
The International Society for
Computers and Their Applications - ISCA

ISBN: 1-880843-22-6

Solving Block-Structured Linear Systems on a Power Challenge 8000

M. Paprzycki
Department of Computer Science
and Statistics
University of Southern Mississippi
Hattiesburg, MS 39406, USA

P. Yalamov
Center of Applied Mathematics
and Informatics
University of Russe
7017 Russe, Bulgaria

Abstract

Discretization of a large number of mathematical problems leads to block-structured matrices. Operations on these matrices become the most expensive part of the solution of the original problem. To achieve an efficient solution, a level 3 BLAS based library of subroutines is proposed and its performance on an SGI Power Challenge 8000 is illustrated.

1 Introduction

A number of mathematical problems give rise to block bidiagonal (*BBD*), block tridiagonal (*BTD*), almost block diagonal and other block-structured linear systems [1,6,8,10]. The need for efficiency in dealing with such matrices becomes particularly transparent when (a) a non-linear problem is being solved and thus a block-structured matrix has to be factorized in each step of an iterative scheme, and (b) when a tearing-type strategy is applied to a very large block-structured matrix to achieve parallelization of the solution process [1,2,5]. In both cases the efficiency of the matrix operations becomes the bottleneck of the solution process.

In this note we present the efficiency of a level 3 BLAS based library of subroutines [3] performing matrix multiplication, matrix factorization and a back-solve for block-structured matrices. Due to lack of space we will limit our attention to block bidiagonal and block tridiagonal matrices and to the SGI Power Challenge 8000 computer (timing results were collected using the *dtime* utility and each result is an average of multiple runs). Results of earlier experiments on the Cray J-916 computer can be found in [7,9]. We will try do address one basic question: Should block structured matrices be represented in the block-structured form, or should they be represented as banded matrices.

2 Matrix-Vector Multiplication

It is easy to visualize how the matrix-vector and the matrix-matrix multiplication of block bidiagonal and block tridiagonal matrices can be realized by a sequence of calls to the level 3 BLAS matrix multiplication routine *_GEMM*. The proposed library allows both straightforward multiplication and a multiplication by a transposed matrix. The latter is achieved by appropriately applying the *transpose* option of the *_GEMM*. The performance of the library routines will be compared with the level 2 BLAS routine *_GBMV* which performs a matrix-vector multiplication for a banded matrix. Let us assume that *A* is a *BTD* (or a *BBD*) matrix consisting of *k* blocks of size *n*. In the library routines it will be represented as a collection of *blocks* stored in a column-oriented fashion. The same *BTD* (*BBD*) matrix can be represented as a banded matrix. For the purpose of the *_GBMV* routine it would be stored as a collection of *vectors* representing the main diagonal and *2n-1* sub-diagonals and *2n-1* super-diagonals (in case of the *BBD* matrix we would have *n-1* sub-diagonals and *2n-1* super-diagonals). This difference in storage scheme had a significant effect on performance on the Cray vector-computer. Although the banded representation stores elements that are outside of the original matrix, the long vectors allowed the banded storage based *_GBMV* to outperform the block-oriented routines for approximately n<30 (see [9] for more details).

In the first series of experiments we compare the performance of the block-oriented and banded matrix-vector multiplication. In Figure 1 we present the ratio of time of the banded and block-oriented routines for the increasing block size *n* for *BTD* and *BBD* matrices and their transposes (denoted by (T)). In this, and the remaining figures, the results presented are for k = 400 blocks. We have found that the number of blocks *k* does not affect the performance.

The results indicate that for small blocks the banded representation should be used. It is only for n>10 when the blocked approach outperforms the banded one. The difference in time is especially visible for the medium-size blocks, and for the largest blocks it tends to decrease which is likely to be an effect of the cache management. It can be, however, predicted that for very large blocks, this difference will increase again as the banded approach will perform a number of additional unnecessary operations on the zero elements that do not belong to the original matrix.

The situation is different for the transposed matrices. Here, the banded approach is much more efficient. As the block size *n* is increasing the blocked approach seems to be slowly gaining and it can be predicted that for large blocks it should be the approach of choice.

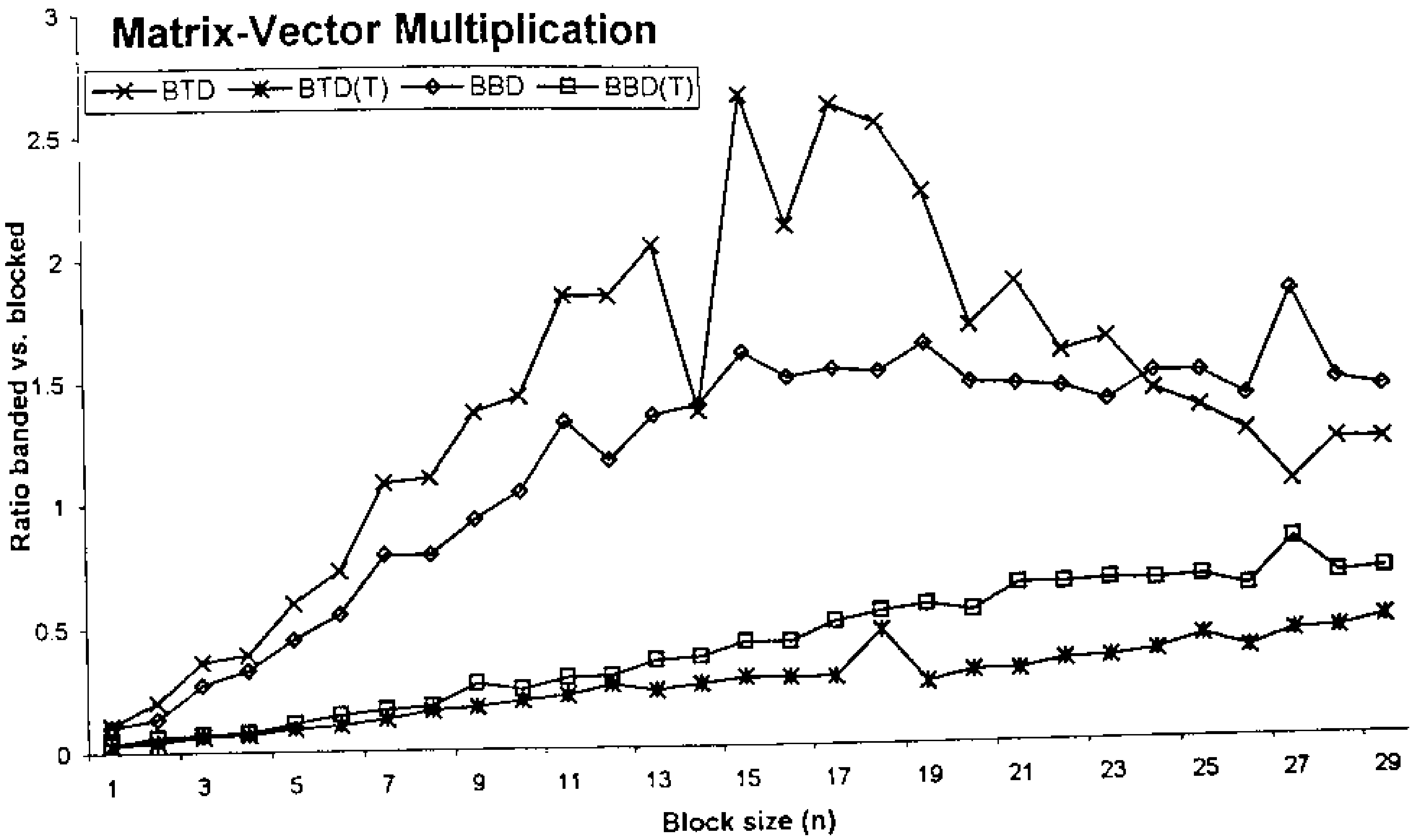

Figure 1. Matrix-vector multiplication banded vs. blocked approach; (T) denotes transpose.

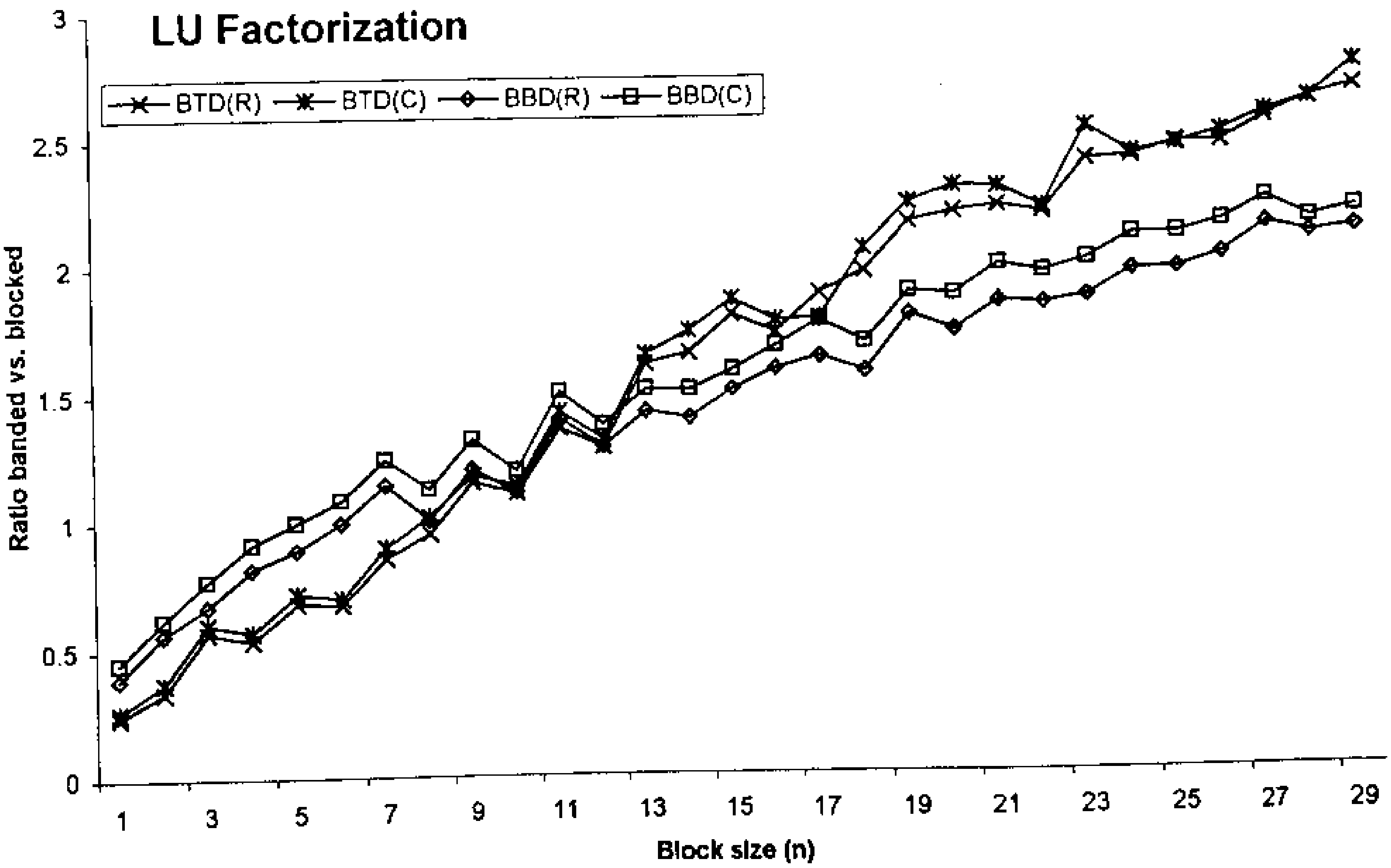

Figure 2. Matrix factorization banded vs. blocked approach; (R) denotes row pivoting, (C) denotes column pivoting.

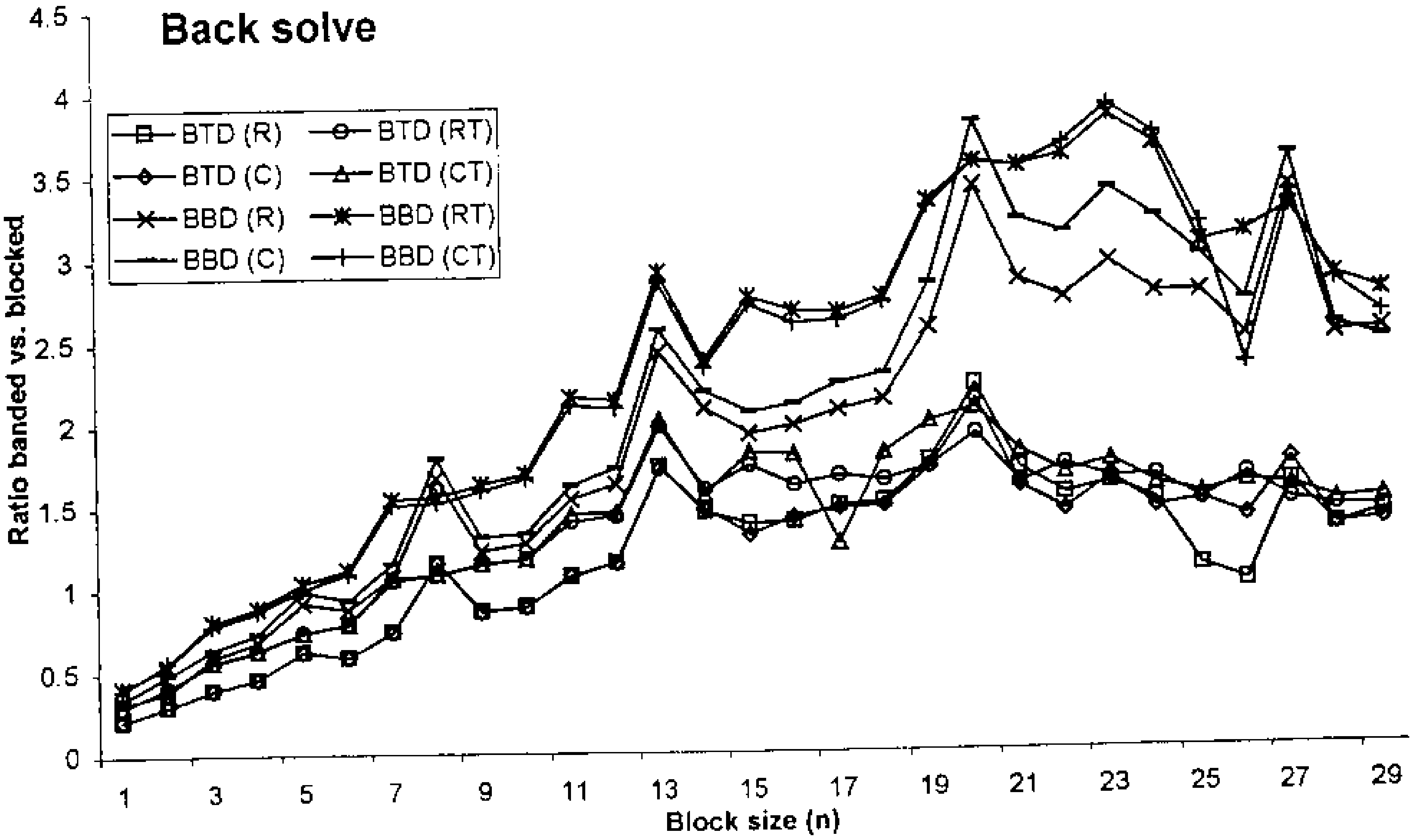

Figure 3. Back Substitution banded vs. blocked approach

3 Matrix factorization

In the second series of experiments we have studied the performance of the LU factorization routines. Here the question of a fair comparison needs to be addressed. The block-algorithms perform the LU factorization with limited pivoting, where the pivot element is sought only in the *middle-block*. This result follows the strategy proposed originally by Varah in [10]. In the same paper the detailed result concerning the stability of this approach can be found. The factorization routine for the banded matrices from the LAPACK library *_GBTRF* uses pivoting and row interchanges. This means that a fill-in of the size of the upper bandwidth is generated (and additional arithmetical operations performed).

We believe that since pivoting is also performed by the block-oriented algorithms it is fair to make a straightforward comparison between the two approaches. It is assumed that the additional cost of the banded solver is part of the price one has to pay when using a black-box library software. Finally, in both cases of *BBD* and *BTD* matrices we have combined pivoting with row (denoted as (R)) and column (denoted as (C)) interchanges to see if there is a performance difference caused by these two approaches.

The results are much more favorable for the blocked approach than in the case of matrix-vector multiplication. Already for n>9 the block-oriented approaches outperform the banded one. In addition, the performance gain increases as the block size n increases. It is slightly larger for the *BTD* matrices than for the *BBD* matrices and in both cases there is almost no difference between the column and row pivoting strategies. Only for the *BBD* matrices the column pivoting slightly outperforms the row pivoting.

4 Back substitution

In the final series of experiments we have compared the performance of the back solvers for a single right hand side for the matrices decomposed by the factorizing routines described above. In Figure 3 we present the ratio of times used by the level 2 BLAS based banded solver *_GBTRS* and the block-oriented back-solvers for the increasing block size n. Since the row and column pivoting based factorizations have been implemented there are 4 back solvers available: row pivoting (R), row pivoting transposed (RT), column pivoting (C), column pivoting transposed (CT). They are compared with the banded and banded transposed solvers as appropriate.

As in previous cases for n>10 the block-oriented solvers start to outperform the banded approach. Surprisingly, the *BBD* solver creates a more substantial performance gain than the *BTD* solver. There seems to be no particular difference between the four solvers for each blocked matrix; in particular there is no difference between the transposed and the non-transposed solvers. Just as in the case of matrix-vector multiplication, as the value of n increases the difference between the performance of the banded and the block-oriented solvers slowly disappears. Again, it can be predicted that for large values of n the difference will start increasing as the banded solver performs unnecessary arithmetical operations.

5 Concluding remarks

In this note we have compared the performance of the banded and the block-oriented approaches to the basic operations (multiplication, factorization and back-substitution) on block-structured matrices. We have found that on the SGI Power Challenge 8000 RISC-based supercomputer *for small blocks $n<10$ the banded approach is superior to the block-oriented approach for all operations and should be the method of choice in the computational practice*. As the block size increases the blocked approach outperforms the banded one. This is especially true for the matrix factorization, the most time consuming operation of the investigated. These results differ slightly from those reported for the Cray vector-processors where the banded approach has outperformed the blocked approaches for $n < 30$ for matrix-vector multiplication and a straightforward back-substitution. In the case of matrix factorization the results on both computers were similar. Finally, on the Cray the blocked and banded transposed back-substitution runs equally fast [10]. These results present an interesting challenge to the designers of codes like e.g. COLNEW [4] which relies heavily on block structured linear algebra. In the near future we will expand our investigation and consider the almost block diagonal matrices and other RISC based high performance computers (in particular the SGI Power Challenge 10000 and the Convex Exemplar).

Acknowledgements

Computer time grant from NCSA at Urbana is kindly acknowledged. The research has been initiated by a COBASE grant from the National Research Council. The second author was supported by Grant MM-434/94 from the National Research Fund of the Bulgarian Ministry of Education and Science.

References

[1] P. Amodio, T. Politi, M. Paprzycki, "Survey of Parallel Algorithms for Block Bidiagonal Linear Systems," *Journal of Computers in Mathematics Applications*, Vol. 31(7), pp. 111-127, 1996

[2] P. Amodio, T. Politi, M. Paprzycki, "Solving Block Bidiagonal Linear Systems on Distributed Memory Compters," *Proceedings of the Seventh International Conference on Parallel and Distributed Computing Systems*, ISCA, Raleigh, NC. pp. 812-815, 1994

[3] 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

[4] U. Ascher, J. Christiansen, R. D. Russell, "*Numerical Solution of Boundary Value Problems for Ordinary Differential Equations*," Prentice-Hall, New York, 1988

[5] M. Berry, A. Sameh, "Multiprocessor Schemes for Solving Block Tridigonal Linear Systems," *The International Journal of Supercomputing Applications*, Vol. 2, pp. 37-57, 1988

[6] C. Cyphers, M. Paprzycki, A. Karageorghis, "High Performance Solution of Partial Differential Equations Discretized Using a Chebyshev Spectral Collocation Method," *Journal of Computational and Applied Mathematics*, Vol. 66(1), pp. 71-80, 1996

[7] M. Paprzycki, C. Cyphers, "Level 3 BLAS Based Library for Block Tridiagonal Matrices," in: S. Elaydi, et. al. (eds.), *Advances in Difference Equations*, Gordon and Breach, Amsterdam, pp. 491-498, 1997

[8] M. Paprzycki, I. Gladwell, "Solving Almost Block Diagonal Systems Using Level 3 BLAS," *Proceedings of The Fifth SIAM Conference on Parallel Processing for Scientific Computing*, SIAM, Philadelphia, pp. 52-62, 1992

[9] M. Paprzycki, A. Karageorghis, C. Cyphers, "Solving Structured Matrix Problems on a Cray Vector-Computer," Technical Report, University of Cyprus, submitted for publication

[10] J. Varah, "On the Solution of Block-Tridiagonal Systems Arising from Certain Finite-Difference Equations," *Mathematics of Computation*, Vol. 26, pp. 859-868, 1972