PROCEEDINGS OF THE TENTH ANNUAL

CONFERENCE ON

APPLIED MATHEMATICS

University of Central Oklahoma
Edmond, Oklahoma
February 4-5, 1994

Optimizing Performance of Basic Linear Algebraic Subroutines on DEC Computers

MARCIN PAPRZYCKI

CLIFF CYPHERS

Department of Mathematics and Computer Science

University of Texas of the Permian Basin

Odessa, TX 79762

(915) 367-2244

paprzycki_m@gusher.pb.utexas.edu

cliffc@tenet.edu

ABSTRACT: Many recently published papers study optimizing techniques for vector-supercomputers. The aim of this paper is to investigate the feasibility of these techniques on other computer architectures. We will study the effects of loop unrolling and blocking on the DEC 5000/200 RISC architecture and VAX 4000 mainframe.

1. INTRODUCTION

Recent advances in the development of computer architectures facilitated a radical increase in the size of the problems to be solved. To be able to satisfy the needs of computationally intensive applications not only hardware but also software must be optimized. Numerical linear algebra is one of the areas where substantial advances in knowledge about code optimization for modern architectures have been made. This can be partially explained by the fact that there exists a large number of problems (typically involving large dense matrices) which require $O(n^3)$ operations so that any time savings are crucial. The study of problems involving large dense matrices is at the center of the LAPACK project [1] and has led to establishing level 1, 2 and 3 BLAS kernels [4, 5, 10] as well as to the study of a number of optimizing techniques. A large number of recently published papers examine such techniques for vector-supercomputers [2, 3, 6, 8, 9]. The aim of this paper is to investigate the feasibility of these techniques on more standard computer architectures. Two optimizing techniques (loop unrolling and blocking) applied to vector addition, dot product, matrix-vector and matrix-matrix multiplication will be considered. It will be shown that such techniques can be successful and lead to substantial performance gains.

We have experimented on two computer architectures: VAX 4000-100 mainframe with 64 Mbytes of main memory and DEC 5000/200 workstation with 32 Mbytes of main memory. All programs were written in Fortran 77. The timings on the VAX 4000 were obtained using the system timer (library routines LIB$INIT_TIMER and LIB$SHOW_TIMER), which provides information about the actual CPU time used.

Times on the DEC 5000/200 were measured on an empty machine using the Unix function *seconds*. All results presented in this paper are averages of multiple runs. In all cases, the performance is represented by dividing the number of floating point arithmetical operations necessary to calculate the result by the CPU time used to calculate the result, and scaled by 10^{-6} (this performance measure can be interpreted as approximating MFlops).

Section 2 concerns vector-vector operations, matrix-vector operations are considered in section 3, section 4 deals with matrix-matrix operations.

2. Vector-Vector Operations

Loop unrolling is a standard optimizing technique which can be applied to vector-vector operations [3, 6]. The level 1 BLAS standard [10] defines several basic vector-vector operations. The first operation we would like to consider is the vector update of the form (this is the AXPY operation in the level 1 BLAS definition):

$$x = x + \alpha y$$

where $x, y \in R^n$ and α is a scalar. A generic Fortran loop that would perform this operation has the form

```
      DO 10 I = 1,N
10          X(I) = X(I) + α*Y(I).
```

As suggested in [3, 6] this loop can be unrolled to varying depths (the example shows unrolling to level 4; it is assumed that care is taken of the possible uneven vector parts):

```
      DO 10 I = 1,N,4
            X(I)   = X(I)   + α*Y(I)
            X(I+1) = X(I+1) + α*Y(I+1)
            X(I+2) = X(I+2) + α*Y(I+2)
            X(I+3) = X(I+3) + α*Y(I+3)
10    CONTINUE
```

By improving the register management, such unrolling leads to substantial performance gains on the Cray supercomputers [3].

The second operation we would like to consider is dot-product calculation (operation DOT from the level 1 BLAS definition). This operation has the form:

$$d = \sum_{i=1}^{N} x_i \, y_i$$

where $x, y \in R^n$. A generic Fortran loop that performs this operation has the form:

```
      DO 10 I = 1,N
10          TEMP = TEMP + X(I)*Y(I).
```

The unrolling of this loop leads to substantial performance gains e.g. on an IBM 3900 [3]. Unrolled to level 4 this loop takes the form (again assuming an appropriate treatment of uneven parts):

```
      DO 10 I = 1,N,4
            TEMP = TEMP + X(I)*Y(I)
                        + X(I+1)*Y(I+1)
                        + X(I+2)*Y(I+2)
                        + X(I+3)*Y(I+3)
10    CONTINUE
```

The results of experiments on the DEC 5000 and VAX 4000 for both operations and various depths of unrolling are summarized in Table 1 (the vector length 40,000). For both operations we have also experimented with other vector sizes and the results were similar.

For the AXPY operation the results on the VAX 4000 are not surprising. From this and other experiments we have learned that a performance gain of approximately 7% from unrolling can be observed. The best results for all vector sizes were for unrolling approximately to the level of 8. At the same time a surprising result can be observed on the DEC 5000, where any attempt at unrolling leads to performance losses.

For the DOT operation we can observe a definite performance improvement for the unrolled versions on both machines. Performance gains on the VAX 4000 are

Unrolling depth	AXPY		DOT	
	VAX 4000	DEC 5000/200	VAX 4000	DEC 5000/200
1	2.85	1.99	3.18	2.17
2	2.86	1.99	2.73	2.22
3	2.83	1.89	3.15	2.26
4	2.86	1.93	3.23	2.24
5	2.99	1.92	3.22	2.28
6	3.02	1.94	3.18	2.27
7	3.03	1.82	3.27	2.29
8	3.05	1.89	3.30	2.29
9	3.03	1.91	3.29	2.26
10	3.02	1.91	3.30	2.30
11	3.03	1.86	3.32	2.26

Table 1. Performance of the unrolled AXPY and DOT operations.

approximately 3.7% for all vector sizes experimented with, and were observed for unrolling approximately to the level of 10 or 11. There is also a definite performance gain from using loop unrolling on the DEC 5000. For all of our experiments the

performance gain of up to 6% was observed and the best results were for unrolling approximately up to the level of 10.

3. Matrix-Vector Operations

As an example of the matrix-vector operation we have used matrix-vector multiplication (GEMV operation from level 2 BLAS [5]), a generic Fortran program of which has the form:

```
      DO 10 J = 1,N
            DO 10 I = 1,N
                  Y(I) = Y(I) + A(I,J)*X(J)
10          CONTINUE
```

where $x, y \in R^n$ and $A \in R^{n \times n}$. This time two directions of unrolling are possible: *to the right* (for DOT-type unrolling) and *down* (for AXPY-type unrolling). After unrolling to the level of 2 to the right and to the level of 3 down the above loop becomes (again it is assumed that uneven parts have been taken care of):

```
      DO 10 I = 1,N,3
            DO 10 J = 1,N,2
                  Y(I)   = Y(I)   + A(I,J)*X(J)   + A(I,J+1)*X(J+1)
                  Y(I+1) = Y(I+1) + A(I+1,J)*X(J) + A(I+1,J+1)*X(J+1)
                  Y(I+2) = Y(I+2) + A(I+2,J)*X(J) + A(I+2,J+1)*X(J+1)
10          CONTINUE
```

Figures 2 and 3 present the performance results for both computers for various levels of unrolling in both directions and matrix size 400. Similar results were observed also for other matrix sizes.

It can be observed that there is a significant difference between the unrolled and the not unrolled implementations. As was suggested by the results of vector-vector operations, the best results are observed for the unrolling to the right (analogous to the

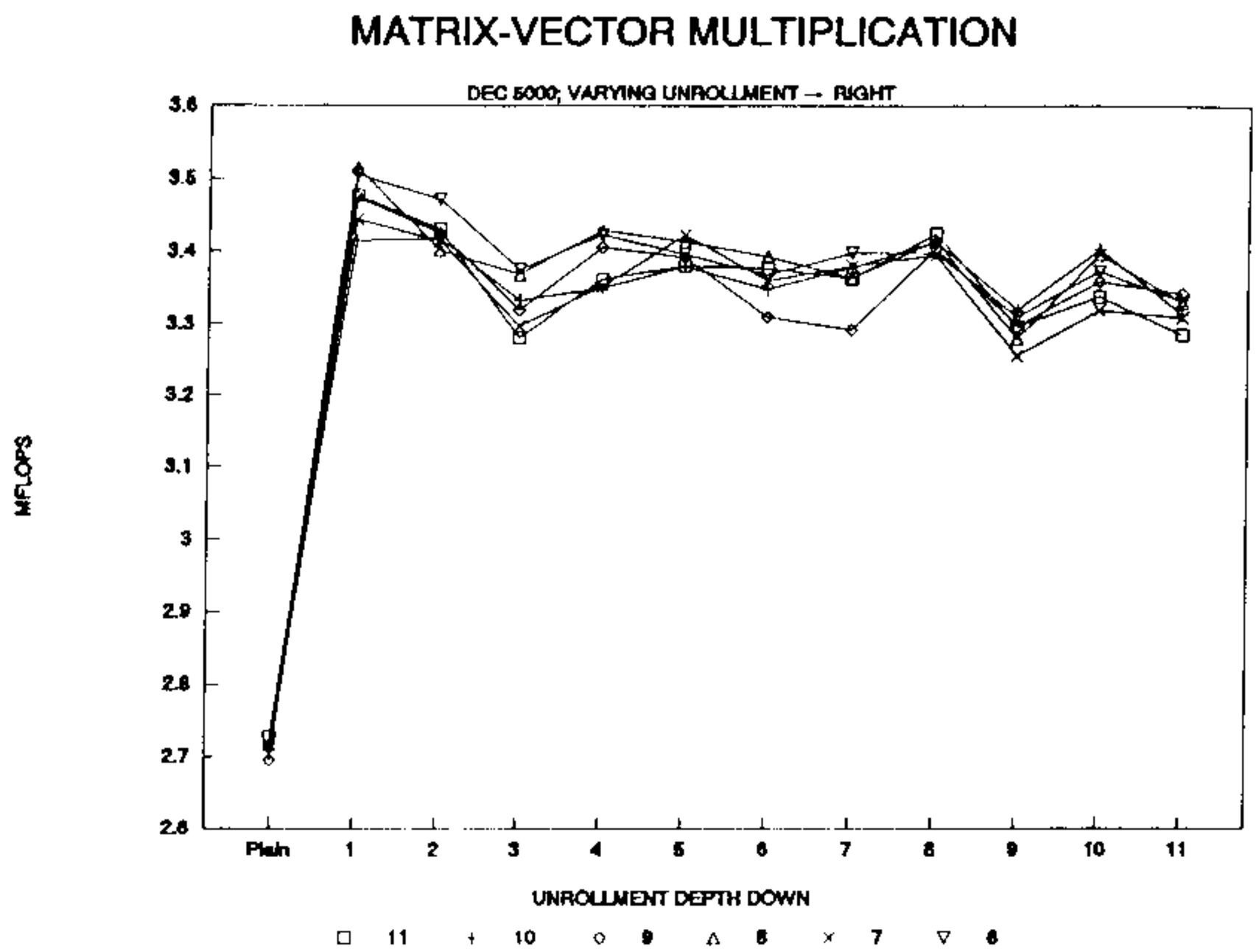

Figure 1. Performance of the unrolled GEMV; DEC 5000/200.

DOT operation) to the level of 9, 10 or 11. Overall performance gain from unrolling is approximately 30%.

It can be observed that a gain of approximately 23% can be achieved when matrix-vector multiplication is unrolled by 10 down and 1 or 3 to the right. This result is in agreement with what we have observed for vector-vector operations, where a more significant gain was observed for AXPY operations.

4. Matrix-Matrix Operations

Basic matrix-matrix operations are defined by the level 3 BLAS standard [4]. A typical example of such an operation is matrix multiplication. Its generic form

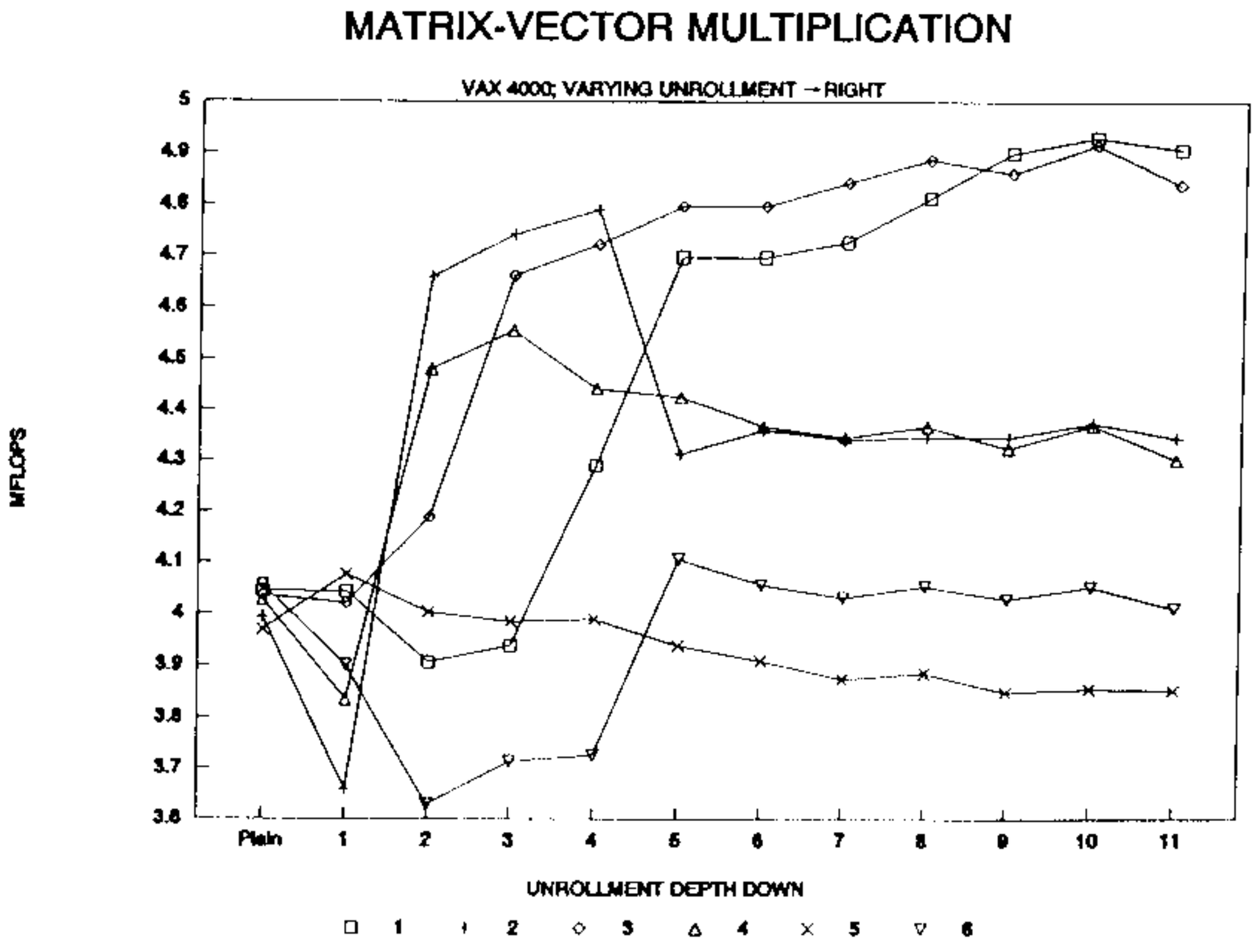

Figure 2. Performance of the unrolled GEMV; VAX 4000.

(presented here is a column oriented form which is the fastest for Fortran [6]) is:

```
DO 10 J = 1,N
        DO 10 K = 1,N
                DO 10 I = 1,N
                        C(I,J) = C(I,J) + A(I,K)*B(K,J)
10      CONTINUE
```

where $A, B, C \epsilon R^{n \times n}$. When multiplying matrices there are several ways of optimizing performance. The innermost do-loop represents a number of vector updates (AXPY operations), so we can replace it by a series of calls to the optimized AXPY routine. The two innermost do-loops together represent a sequence of matrix-vector multiplications, so they can be replaced by a call to the optimized GEMV routine. There is one more optimizing technique — blocking. In this case we attempt to reduce

data movement by performing a number of operations on a block of matrix C kept in cache memory [7]. This approach is illustrated in Figure 3. In this case the operations between the blocks are performed by utilizing the most efficient GEMV code.

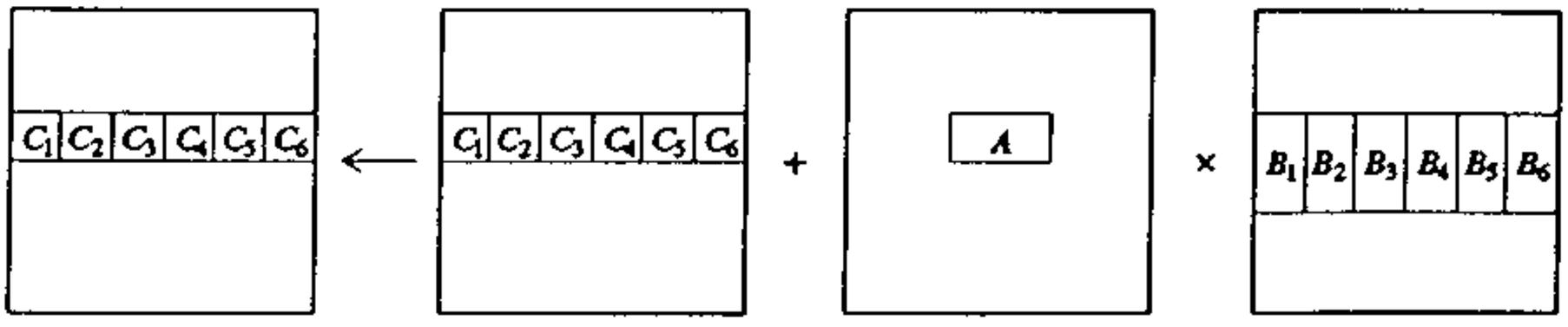

Figure 3. Blocking of a matrix-matrix operation.

The results of experiments using all of the above mentioned techniques for various matrix sizes are summarized in Table 2 (for VAX 4000) and Table 3 for DEC 5000/200 In both cases, we have experimented with a variety of blocksizes and found that the blocksize should be a multiplicity of the level of the unrolling of the GEMV routine. The optimal blocksize is between 40 and 60 and varies for various matrix sizes. The difference between performance of blocked codes for blocksizes in this range is approximately 1-3%. The results in Tables 2 and 3 are for blocksize 40.

	JKI	JK-AXPY	J-GEMV	BLOCKED
100	4.46	5.99	6.81	8.33
150	3.91	5.07	5.56	7.87
200	3.82	4.95	5.32	7.86
250	3.72	4.74	5.23	7.42

Table 2. Matrix-matrix multiplication, VAX 4000, performance in MFlops

For large matrices, when the optimized AXPY operation is used inside the matrix multiplication a performance gain of approximately 27% can be observed. When the optimized GEMV operation replaces the two innermost do-loops the performance gain over the plain code is approximately 40%. For blocking, when used in matrix-matrix multiplication, the performance gain over the plain method is approximately 99%.

	JKI	JK-AXPY	J-GEMV	BLOCKED
100	3.90	4.06	4.97	6.29
150	2.77	2.81	3.40	6.57
200	2.79	2.82	3.40	6.63
250	2.78	2.81	3.41	6.50

Table 3. Matrix-matrix multiplication, DEC 5000/200, performance in MFlops

As one could expect application of optimized AXPY does not lead to major performance gains. Optimized GEMV leads (for large matrices) to the performance gains of approximately 22%. When an optimized blocked code is applied the performance gain over the plain code reaches 133%.

5. Conclusions

In this paper we have discussed the possible performance gains when basic optimizing techniques are applied to simple linear algebraic operations. It is shown that such techniques lead to substantial gains not only on supercomputers. It is shown, that the biggest performance gain (even more than 100%) can be obtained when blocking is combined with optimal matrix-vector and vector-vector operations.

REFERENCES

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

2. Anderson, E., Dongarra, J., Evaluating Block Algorithm Variants in LAPACK, in: Dongarra, J., Messina, P., Sorensen, D.C., Voigt, R.G., (eds.) *Parallel Processing for Scientific Computing*, SIAM, Philadelphia, 1989

3. Dayde, M.J., Duff, I.S., Level 3 BLAS in LU Factorization on Cray-2, ETA-10P and IBM 3090-200/VF, *The International Journal of Supercomputer Applications*, **3** (2), 1989, 40-70

4. Dongarra, J.J., Du Croz, J., Duff, I., Hammarling, S., "A Set of Level 3 Basic Linear Algebra Subprograms," *Technical Report ANL-MCS-TM57*, Argonne National Laboratory, 1988

5. Dongarra, J.J., Du Croz, J., Hammarling, S., Hanson, R.J., An Extended Set of FORTRAN Basic Linear Algebra Subprograms, *ACM Trans. on Mathematical Software*, **14** (1), 1988, 1-17

6. Dongarra, J.J., Gustavson, F.G., and Karp, A., Implementing Linear Algebra Algorithms for Dense Matrices on a Vector Pipeline Machine, *SIAM Review*, **26**, 1984, 91-112

7. Dongarra, J. J., Mayes, P., Radicati, G., The IBM RISC System/6000 and linear algebra operations, *SuperComputer*, **8** (4), 1991, 15-30

8. Gallivan, K., Jalby, W., Meier, U., Sameh, A., Impact of Hierarchical Memory Systems on Linear Algebra Algorithm Design, *The International Journal of Supercomputer Applications*, **2** (1), 1988, 12-46

9. Gallivan, K., Plemmons, J.R., Sameh, H.A., Parallel Algorithms for Dense Linear Algebra Computations, *SIAM Review*, **32** (1), 1990, 54-135

10. Lawson, C.L., Hanson, R.J., Kincaid, D.R., and Krogh, F.T., Basic Linear Algebra Subprograms for FORTRAN Usage, *ACM Trans. on Mathematical Software*, **5** (3), 1979, 306-323