The University of Texas System

CENTER FOR HIGH PERFORMANCE COMPUTING

Balcones Research Center, Commons Building 1.154, 10100 Burnet Road, Austin, Texas 78758-4497 • (512) 471-2472

CHPC Newsletter, Vol. 6, No. 6 3 June 1991

Multiplying Matrices on the Cray - Practical Considerations

Marcin Paprzycki and Cliff Cyphers

Department of Mathematics and Computer Science
The University of Texas of the Permian Basin

This article is written to provide users of the Cray Y-MP8/864 computer with some practical guidelines about the best ways of multiplying matrices. Three major issues will be addressed: the impact of the memory bank conflicts, the benefits of using level 3 BLAS routines, and the advantages and limitations of Strassen's algorithm. Our considerations are restricted to single-processor applications.

Matrix multiplication is a very simple process. The following Fortran code represents the way we do it "by hand" (it is assumed that A, B, and C are NxN matrices, the operation is C = A*B, and initially C = 0):

```
1            do 10 I = 1,N
2                  do 10 J = 1,N
3                       do 10 K = 1,N
4      10                     C(I,J) = C(I,J) + A(I,K)*B(K,J)
```

It is, however, a well known fact [2] that permuting the order of loops leads to six separate implementations of matrix multiplication. The primary difference between these methods is the order in which memory locations are accessed. The method described above (IJK method) is based on dot-product calculation, whereas the JKI method (loops ordered 2-3-1) is based on column-vector updates. Since matrices in Fortran are stored in column major order, the latter method should yield a more efficient code. When updating a column of C, it accesses consecutive memory locations distributed among separate memory banks and thus minimizes the number of possible memory bank conflicts.

[*Continued on page 78*]

Contents

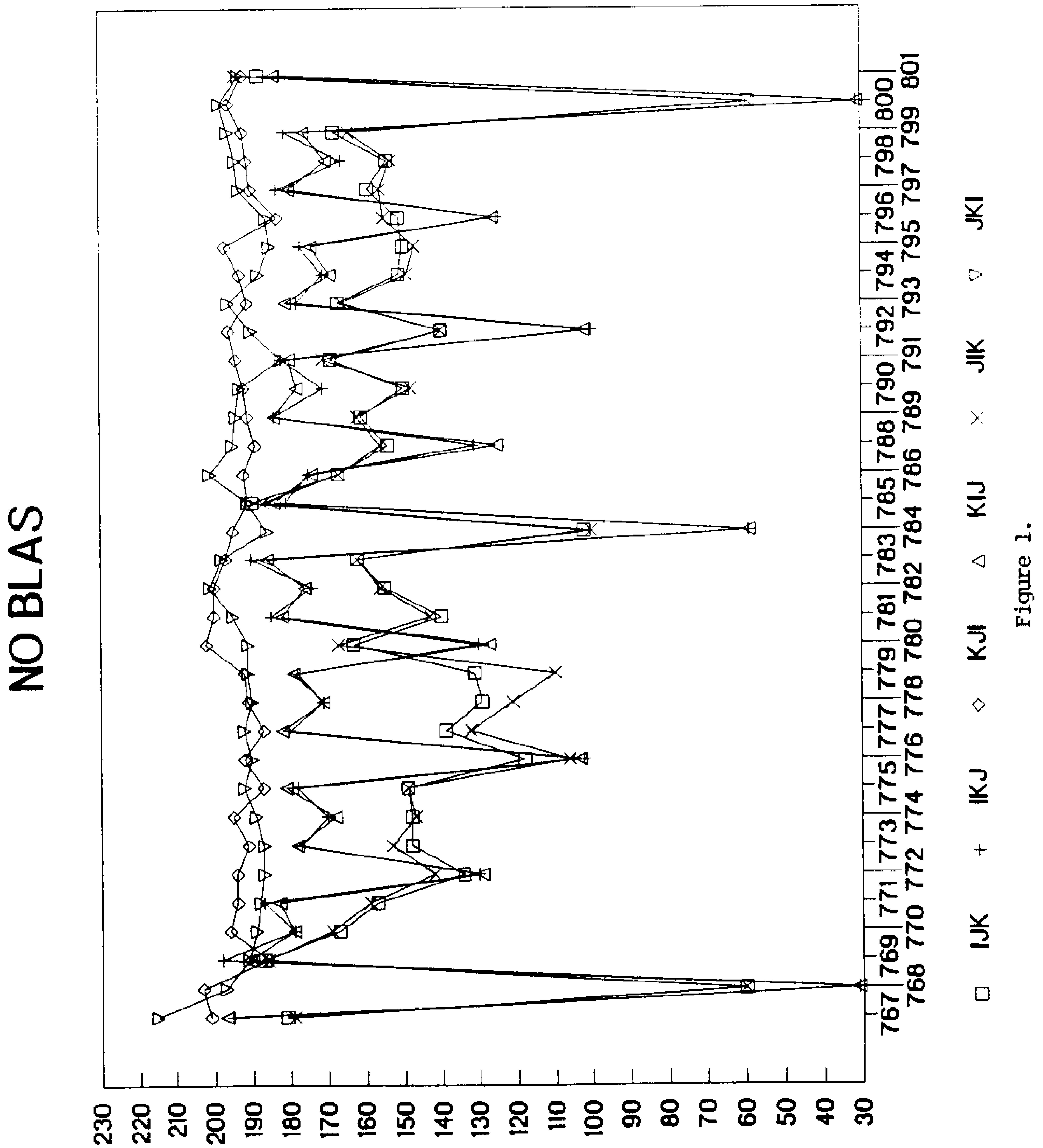
NO BLAS
230
220
210
200
190
180
170
160
150
140
130
120
110
100
90
80
70
60
50
40
30
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
788
789
790
791
792
793
794
795
796
797
798
799
800
801
□ IJK
+ IKJ
◇ KJI
△ KIJ
× JIK
▽ JKI
Figure 1.

Figure 1.

To observe differences in efficiency, we performed experiments with all six methods and matrix sizes from 767 to 801. Our assumption (based on Sewell [7]) was that the worst case performance will occur for the matrix of size 768 as it is a multiple of 256 (the number of memory banks). The results are summarized in Figure 1. Each result is an average of three runs since the perftrace monitor, used to establish the performance characteristics, yielded results that varied up to 5 MFLOPS.

As predicted above, one can observe three performance patterns. They clearly correspond to the innermost do-loop variable and thus to the memory access patterns.

The best performance was achieved by JKI and KJI methods (in the innermost do-loop both update a column of matrix C using a column of matrix A). No effects of memory bank conflicts were observed for these methods.

The worst performance (MFLOP-rate decreased by up to a factor of 7) was observed for IKJ and KIJ methods. In their innermost do-loop, both update a row of matrix C using a row of matrix B. Interestingly, the performance for the matrix size 768 is exactly as bad as for 800. This seems to suggest that the performance degradation can occur for matrices with sizes divisible by 32 rather then by 256. Chris Hempel from Cray Research suggested that this possibly means that, instead of memory bank conflicts, we have here a case of memory section conflicts. It is also noteworthy that the results for these methods are almost symmetrical around 784. This suggests that in the worst case all powers of 2 lead to significant performance decrease.

It should be added that when we replaced the innermost do-loop by calls to level 1 BLAS [5] (routines SAXPY and SDOT) and ran experiments for the same matrix sizes we observed similar behavior as in the no-BLAS case (including the symmetry of the worst case performance). Analogous effects were observed when two inner do-loops were replaced by calls to level 2 BLAS [3] routine SGEMV.

In general, our experiments show that if the proper loop ordering is used, the problems caused by memory access conflicts can be avoided. For the remaining part of this note, we will therefore concentrate on the best possible results.

In [1] the authors concluded that in the case of Gaussian elimination there is a lot to be gained when using level 1 and level 2 BLAS routines. Now we would like to discuss the advantages of using BLAS for matrix multiplication.

As mentioned above, it is possible to replace the innermost do-loop, or two inner do-loops by calls to level 1 or level 2 BLAS, respectively. Cray's Scientific Library also supports the level 3 BLAS [4] that performs the matrix-matrix operation by a call to one routine.

There are three matrix multiplication routines available: a standard SGEMM routine, Cray's matrix multiplication routine MXM, and a special SGEMMS routine that implements Winograd's version of Strassen's iterative matrix multiplication algorithm. The comparison between performances of the best versions of no-BLAS code, level 1 BLAS code, level 2 BLAS code, level 3 BLAS routine SGEMM, and routine MXM for matrices in the range from 100 to 1500 is presented in Figure 2. Because of its nonstandard features, we address the use of SGEMMS below.

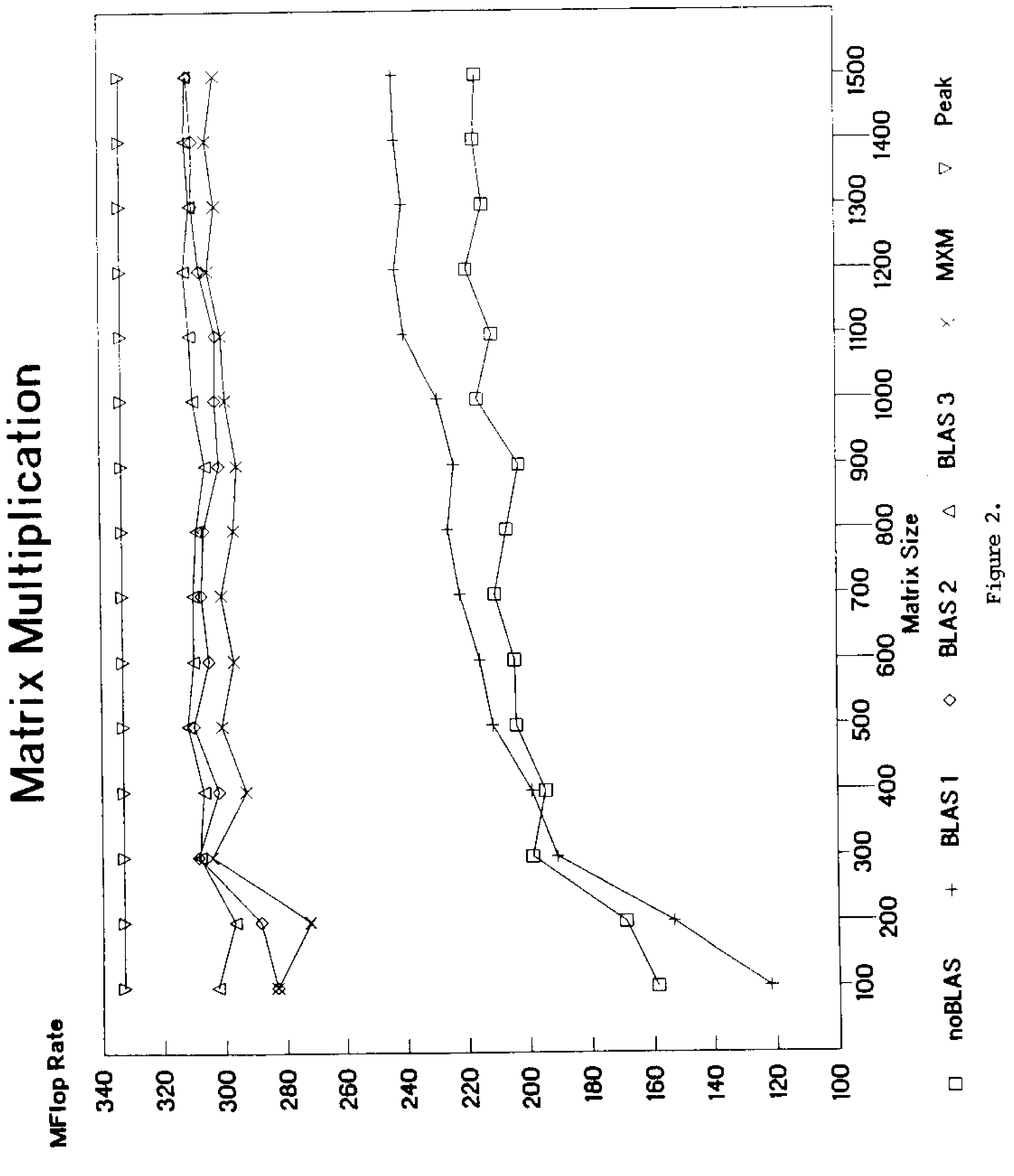
Matrix Multiplication
MFlop Rate
340
320
300
280
260
240
220
200
180
160
140
120
100
100
200
300
400
500
600
700
800
900
1000
1100
1200
1300
1400
1500
Matrix Size
□ noBLAS
+ BLAS 1
◇ BLAS 2
△ BLAS 3
× MXM
▽ Peak
Figure 2.

Figure 2.

The best performance was obtained for the level 3 BLAS routine SGEMM. Interestingly, using level 2 BLAS yields (in most cases) better results then Cray's MXM routine. It is also surprising how close is the performance of the level 2 and level 3 BLAS. This only supports the well-known fact that the level 3 BLAS was intended to achieve the best performance on parallel computers. For the time being, however, only a one-processor version of level 3 BLAS is available. The upgrade to UNICOS version 6.0 and introduction of the LAPACK library will provide fully parallel level 3 BLAS routines.

For large matrices the no-BLAS code reached approximately 66% of the peak performance. Since matrix multiplication is viewed as a perfectly vectorizable operation, the obtained 66% may be treated as the current upper limit of what can be gained when a higher-level programming language and an optimizing compiler are used. The assembly-coded BLAS 2 and BLAS 3 routines reached about 93% of the peak performance.

We repeated our experiments for very small systems. Our results show clearly that for matrices bigger than N=5, SGEMM outperforms all other approaches.

In the final part of this note we would like to address the performance of the SGEMMS routine. It has three important features [6]. Firstly, Strassen's algorithm multiplies matrices using only N**2.8 arithmetic operations in comparison with N**3 for standard matrix multiplication. Consequently, timing rather than MFLOP rate is to be used to compare its performance. Secondly, it is superior to SGEMM only for medium and large matrices. Thirdly, it requires additional workspace of size N*N*2.34. This suggests that there may be a limit on the size of matrices that can be multiplied. Table 1 compares the performance of routines SGEMM and SGEMMS.

Table 1.

Matrix Size	SGEMM time	SGEMMS time
200	0.054	0.047
500	0.804	0.646
800	3.320	2.416
1100	8.583	6.563
1400	17.633	12.133

Table 1. Performance comparison between SGEMM and SGEMMS (time in seconds)

The ratio of the numbers of operations necessary to multiply matrices for both methods is N**.2, and hence the ratio of times spent by both routines should be similar. For all experiments performed, this ratio was satisfied up to a constant. For matrices smaller than 50, SGEMM is superior to SGEMMS. For matrices between 50 and 130, the performance of both routines is almost identical. For matrices bigger than 130, SGEMMS outperforms SGEMM. We were also able to establish that for matrices bigger then 1700, it is impossible to fit matrices A, B, C, and the working space required by SGEMMS in the 16.0 megaword limit on one processor.

Summary:

When a proper order of operations is used, memory-related conflicts can be avoided. Level 3 BLAS routines should be used to multiply matrices. Routine SGEMM is superior for small systems and the only choice for very large systems. Routine SGEMMS should be used for medium and large systems.

References:

1. Cyphers, C., Paprzycki, M., "Gaussian Elimination on the Cray Y-MP8/864," *CHPC Newsletter*, 6 (4), 25 February 1991, 43-47.

2. 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.
3. Dongarra, J. J., Du Croz, J., Hammarling, S., and Hanson, R. J., "An Extended Set of Fortran Basic Linear Algebra Subprograms," *ACM Transactions on Mathematical Software*, 14 (1), 1988, 1-17.
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. Lawson, C. L., Hanson, R. J., Kincaid, D. R., and Krogh, F. T., "Basic Linear Algebra Subprograms for Fortran Usage," *ACM Transactions on Mathematical Software*, 5 (3), 1979, 306-323.
6. Math and Scientific Reference Manual, SR-2081 5.0, Cray Research, Inc.
7. Sewell, G., "Linear Algebra on a Cray Multiprocessor," *CHPC Newsletter*, 6 (5), 1 April 1991, 63-66.