Comparisons of Gaussian Elimination Algorithms on a Cray Y-MP

Marcin Paprzycki
Department of Mathematics and Computer Science
The University of Texas of the Permian Basin
Odessa, Texas 79762

Submitted by Robert J. Plemmons

ABSTRACT

The results of various implementations of Gaussian elimination on full matrices on a single processor Cray Y-MP are presented and discussed. It is shown that when the manufacturer supplied BLAS kernels are used, the difference between the best versions of level 2 BLAS and level 3 BLAS (blocked) implementations is almost negligible. It is suggested that to improve the performance of blocked Gaussian elimination it is possible to utilize Strassen's matrix multiplication algorithm.

INTRODUCTION

We shall consider the solution of a system of linear equations

$$Ax = b,$$

where A is an $N \times N$ real dense matrix, using Gaussian elimination with partial pivoting. A number of recent publications studied the use of BLAS [8, 9, 15] primitives and blocked algorithms [2, 3, 11] on a variety of single processor [6, 7, 14] and parallel [12, 14] computer architectures to solve this problem. In most cases, however, the discussion has been related to FORTRAN BLAS. Since manufacturer provided BLAS kernels can be much more efficient [5, 17], the aim of this paper is to compare the performance of different versions of Gaussian elimination on a one-processor Cray Y-MP using these kernels.

The BLAS (Basic Linear Algebra Subprograms) standard was designed with two goals in mind: first, to allow portability of codes between different machines, and second, to assure the best available quality of performance on a given system. BLAS routines were designed as standard FORTRAN subroutines with a specified order of parameters and a precise description of the operations performed. Recently, computer manufacturers have come to provide BLAS kernels tuned up specifically for their machines. They can be optimized on many levels—from loop unrolling and/or employment of blocked algorithms (both implemented in FORTRAN) to exclusive coding in assembly language. In each case, the calling sequence and the operation performed remain unchanged, assuring full portability. The optimized versions of BLAS routines are much more efficient than their FORTRAN counterparts, as indicated by recent research (e.g. [5, 16, 17]). For example, using FORTRAN BLAS and optimizing features of the FORTRAN compiler yields up to 66% of the theoretical peak performance on a single processor Cray Y-MP. BLAS kernels coded in Cray Assembly Language, on the other hand, can allow one to obtain up to 93% of the theoretical peak performance [16].

We compared the performance of different versions of Gaussian elimination on a single processor Cray Y-MP using manufacturer provided BLAS kernels. Since we used FORTRAN as the programming language, we considered only three (column oriented) implementations out of the six possible versions of Gaussian elimination [10]. We investigated the performance of DOT, GAXPY, and SAXPY versions of Gaussian elimination as described in [2, 5, 17]. We compared the performance of unblocked (level 2 BLAS based) and blocked (level 3 BLAS based) versions of these algorithms. Our aim was to find out which of the versions is most efficient for the computer architecture in question.

2. COLUMN ORIENTED VARIANTS OF BLOCK *LU* DECOMPOSITION

Our discussion of the three possible column oriented variants of Gaussian elimination is based on [5]. We also follow the authors in the use of notation and figures. We omit pivoting and obvious considerations related to it.

Since the considered variants of Gaussian elimination are column oriented, in each step of the process a block of columns will be decomposed, resulting in the calculation of factors L_i^1, L_i^2, and U_i^1 are presented in Figure 1.

We will start the presentation with the DOT version, which is a variant of Crout's decomposition. In the *i*th step, a block of columns of L and a block of rows of U is computed. Considering the partition presented in Figure 2,

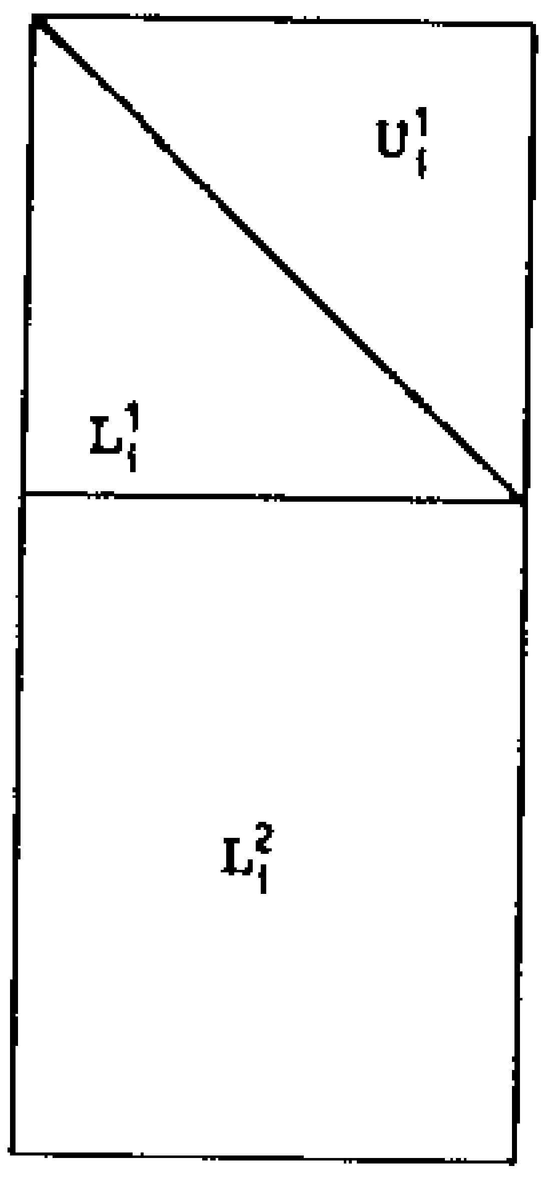

FIG. 1. Decomposition of the block of columns.

one step of the DOT method consists of:

(1) Update of the diagonal and subdiagonal blocks:

$$C_i \leftarrow C_i - A_i B_i.$$

(2) Factorization of a block of columns C_i to obtain factors L_i^1, L_i^2, and U_i^1 (see Figure 1).

(3) Update of a block of rows of U:

$$U_i^2 \leftarrow U_i^2 - A_i^1 E_i.$$

(4) Computation of a block of rows of U:

$$U_i^2 \leftarrow \left(L_i^1\right)^{-1} U_i^2.$$

In each step of the GAXPY version, a block of columns of matrices L and U is calculated. The ith step consists of the following operations (see Figure 3):

(1) Computation of a superdiagonal block of U:

$$U_i^2 \leftarrow \left(L_i'\right)^{-1} U_i^2.$$

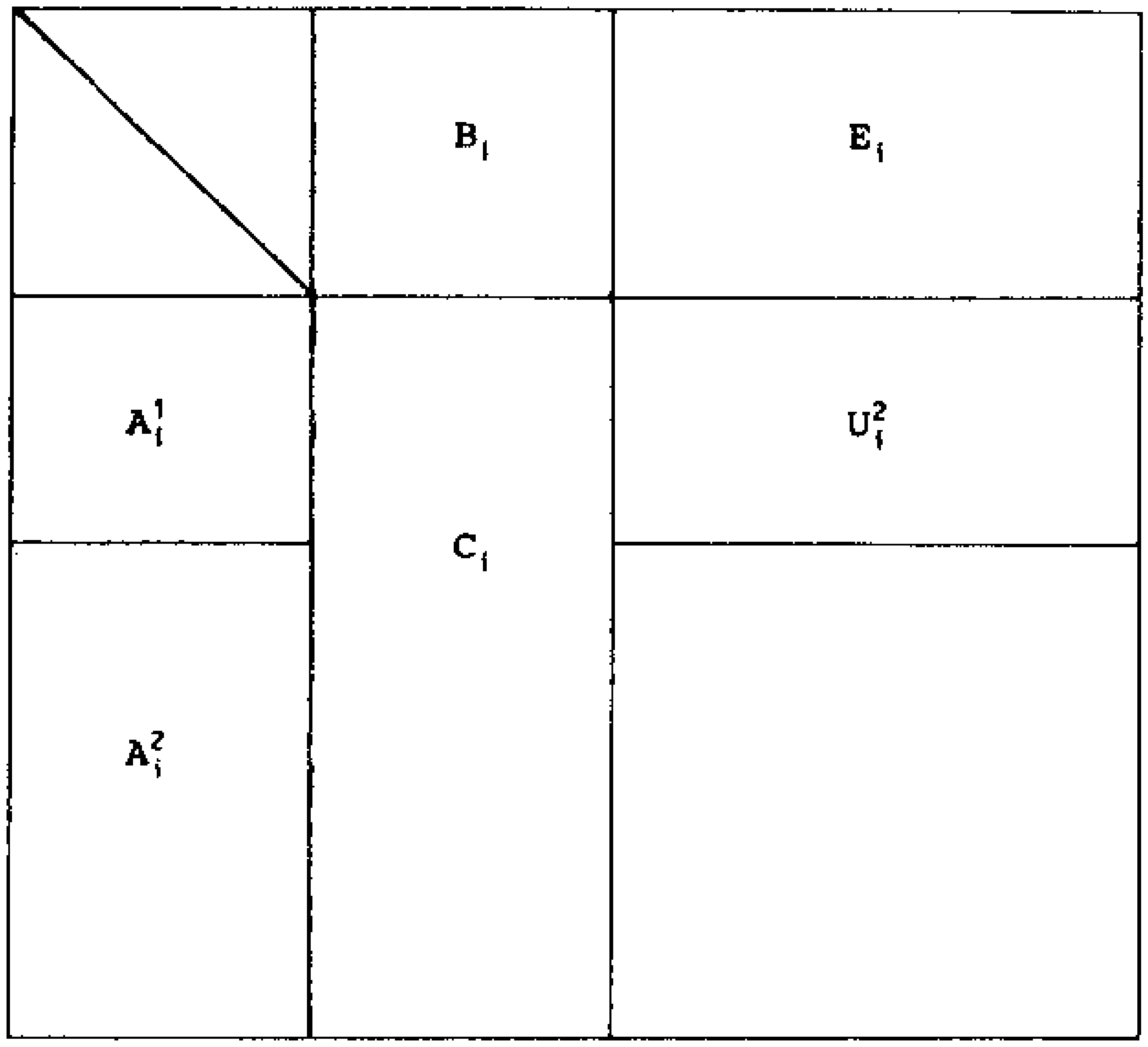

FIG. 2. Partitioning for the DOT variant of Gaussian elimination.

(2) Update of a block to be decomposed:

$$C_i \leftarrow C_i - A_i U_i^2.$$

(3) Factorization of a block of columns C_i to obtain factors L_i^1, L_i^2, and U_i^1 (see Figure 1).

Finally, in the SAXPY variant of the Gaussian elimination, a block of columns of L and a block of rows of U are calculated, and the update is performed on the remaining reduced matrix. In the ith step the following operations are performed (see Figure 4):

(1) Factorization of a block of columns C_i to obtain factors L_i^1, L_i^2, and U_i^1 (see Figure 1).

(2) Computation of a block of rows of U:

$$U_i^2 \leftarrow \left(L_i^1\right)^{-1} U_i^2.$$

(3) Update of the reduced matrix:

$$B_i \leftarrow B_i - L_i^2 U_i^2.$$

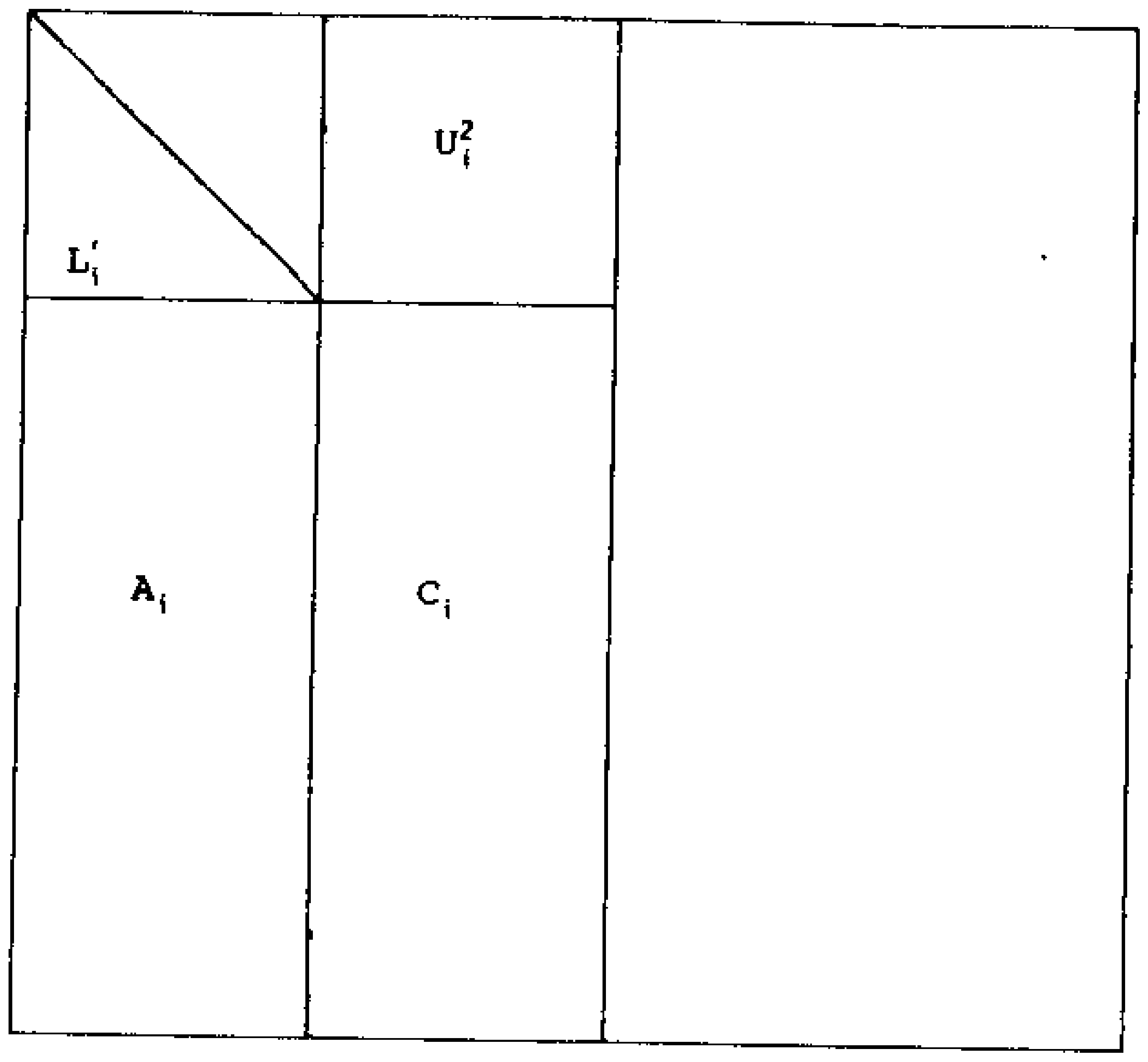

FIG. 3. Partitioning for the GAXPY variant of Gaussian elimination.

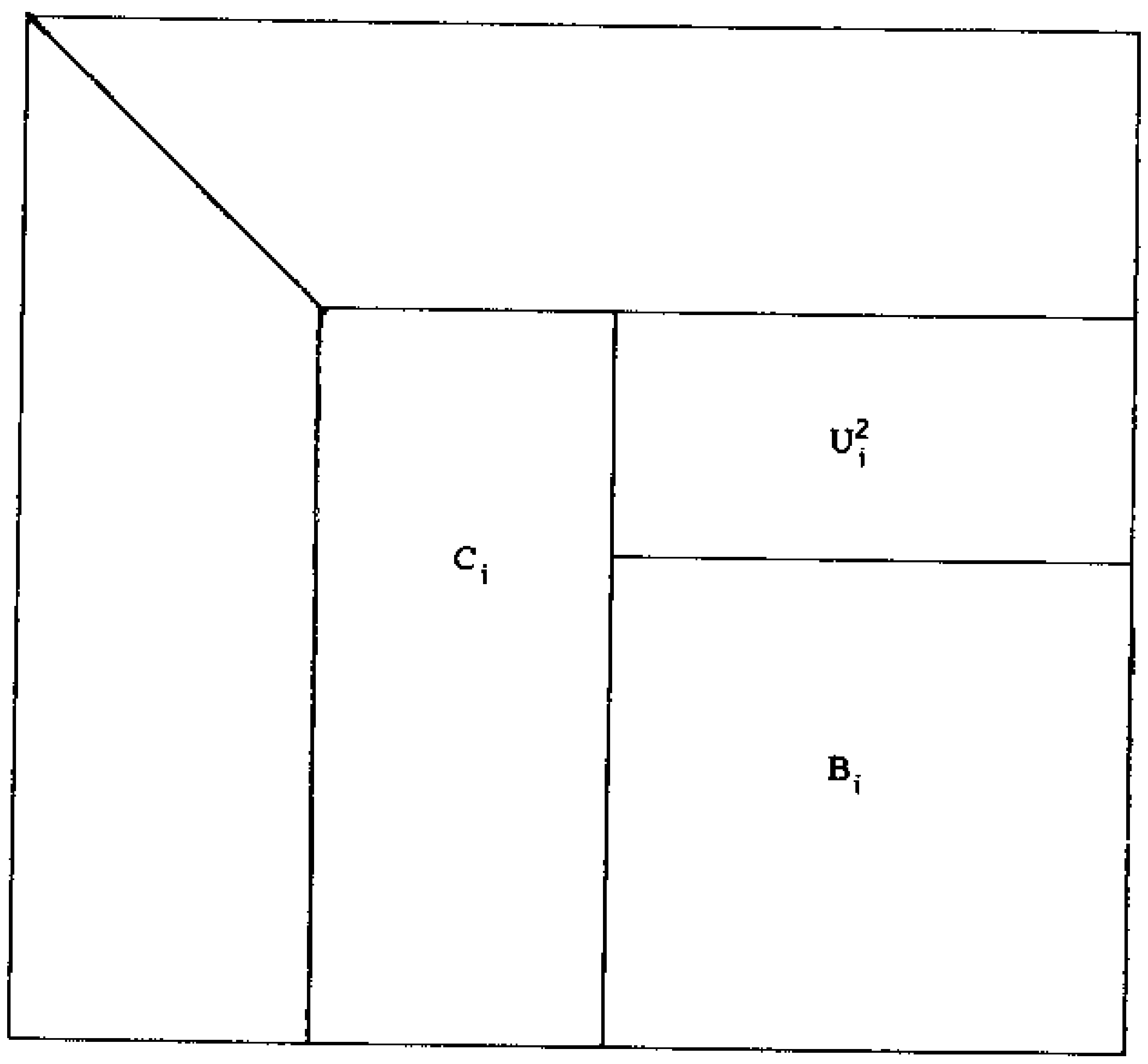

FIG. 4. Partitioning for the SAXPY variant of Gaussian elimination.

3. STRASSEN'S ALGORITHM

In 1969 Strassen [18] showed that it is possible to multiply two matrices of size $N \times N$ with less than $4.7N^{\log_2 7}$ arithmetic operations. Since $\log_2 7 \approx 2.807 < 3$, this method improves asymptotically over the standard matrix multiplication algorithm, which requires $O(N^3)$ operations. Strassen's algorithm is based on the recursive division of matrices into blocks and subsequent performance of block additions, subtractions, and multiplications.

When implemented, Strassen's algorithm involves a tradeoff between a gain in speed and an increase in required storage. In the Cray implementation, the additional work array required by the program was of size $2.34N^2$.

There is one more important consideration, concerning the stability properties of Strassen's algorithm: they are less favorable than those of the conventional matrix multiplication algorithm [13]. The error bounds presented by the author suggest, however, that the expected error growth should not be too serious in computational practice. This conclusion is also backed up by the results of our experiments.

4. LEVEL 3 BLAS BASED ALGORITHMS; IMPLEMENTATION DETAILS

Each of the blocked versions of the Gaussian elimination utilizes three basic block matrix operations: block triangular solve, matrix-matrix product, and reduction of a block of columns. In order to achieve the most effective performance of the blocked code, the most efficient version of each of the three block matrix operations must be used.

For a triangular solve, the existing level 3 BLAS routine STRSM was used. There are, however, three different matrix update routines available on the Cray [4]: the standard, level 3 BLAS SGEMM, the Strassen algorithm based SGEMMS, and the "early Cray" routine MXM. It was shown in [16] that MXM is inferior to both SGEMM and SGEMMS routines for all matrix sizes. SGEMM outperforms SGEMMS for small matrices, but SGEMMS is faster for matrix size larger than 200. We will utilize both routines in the update step of blocked Gaussian elimination and compare their performance.

To reduce a block of columns each of the three unblocked versions of the column oriented elimination can be used. The choice of the most efficient one for a Cray Y-MP was made on experimental basis. All three unblocked versions of Gaussian elimination were coded using calls to appropriate level 1 and level 2 BLAS routines. Since blocks of columns to be reduced in each step of blocked

Gaussian elimination resemble a long strip, we performed our experiments on such long and narrow matrices. Table 1 summarizes the results for matrices with 1024 and 1025 rows when the number of columns (M) varies from 32 to 320. Since memory bank conflicts can cause severe performance deficiencies [16], we have chosen our matrix sizes appropriately. On a Cray Y-MP there are 256 memory banks grouped into 32 memory sections. The matrix size 1024 represents the worst case scenario.

Table 1 shows clearly that there are two versions of unblocked code worth considering. The GAXPY version is most efficient when severe memory section conflicts occur. As Table 1 and other experiments unreported here suggest, the DOT version outperforms the others in the remaining cases.

The DOT version of the unblocked Gaussian elimination was chosen to decompose blocks of columns inside level 3 BLAS implementations. We also performed some experiments with the unblocked GAXPY version, especially for the matrix sizes that caused memory related conflicts (see Section 5 below).

5. NUMERICAL RESULTS

The first series of experiments was designed to compare the performance of the level 2 BLAS and level 3 BLAS based implementations. We ran all three blocked and unblocked versions of Gaussian elimination for $N = 300, 400, \ldots, 1900$. For all experiments the coefficient matrix was generated using the Cray random number generator. To assure accuracy of the presented results each experiment was repeated 50 times; the performance was monitored and averaged by the `perftrace` utility. The blocked codes used SGEMM

TABLE 1
LEVEL 2 BLAS version of Gaussian elimination on matrices of size $1024 \times M$ and $1025 \times M$

	Performance (Mflops)					
	$N = 1024$			$N = 1025$		
M	DOT	GAXPY	SAXPY	DOT	GAXPY	SAXPY
32	225.25	226.86	227.15	231.58	231.58	238.52
64	249.07	253.94	247.42	261.91	258.80	264.18
128	259.01	272.26	259.09	282.87	278.05	278.36
192	253.92	274.77	258.65	290.65	285.10	283.27
256	250.92	280.85	264.02	294.86	288.88	286.05
320	247.07	286.25	268.15	297.50	291.20	287.88

in block updates and unblocked DOT for the decomposition of the block of columns. For blocked codes, a variety of blocksizes was tried, and the best results are presented. The Euclidean norm of error varied from 10^{-10} for small systems to about 10^{-6} for large matrices, which is what one would expect for the Cray single precision arithmetics (the estimated unit roundoff for a Cray Y-MP, established using Moler's algorithm [1; p. 26], is 1.421×10^{-14}). Figure 5 summarizes the results.

It is clear from the results (Figure 5 and Table 1) that the unblocked DOT code is highly sensitive to the memory section conflicts, whereas the performance of the GAXPY version, while not as efficient, is almost unaffected in such situations.

In order to determine precisely the dependency between matrix size and the performance of the unblocked versions of Gaussian elimination, we ran all three unblocked codes for matrix sizes 1025, 1024, . . . , 991. A significant reduction in performance of the unblocked DOT code was observed for matrices of sizes 1024 and 992 when the megaflop rate dropped by approx-

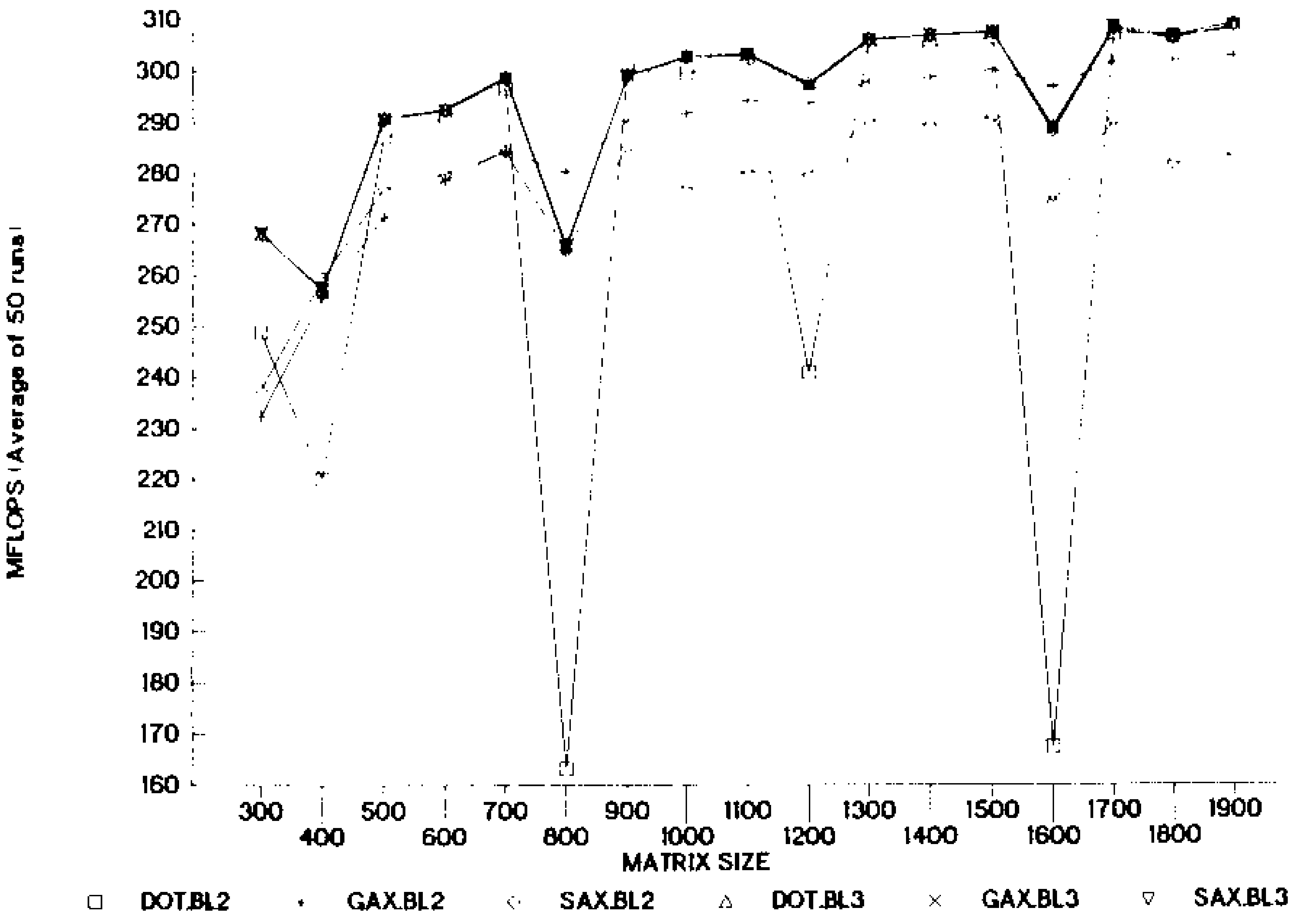

FIG. 5. Comparison between level 2 BLAS and level 3 BLAS based versions of Gaussian elimination; results in megaflops.

imately 50%. Much smaller performance reduction was observed for $N = 1008$—the rate fell approximately 35%. It should be also mentioned that in all cases the GAXPY version outperformed SAXPY. These results match the results from Figure 5. The big dips in performance of the DOT version were observed for $N = 800$ and $N = 1600$; much smaller drops occurred for $N = 400$ and $N = 1200$. In general, due to the memory section conflicts, the unblocked DOT version of Gaussian elimination should be avoided for matrices of sizes divisible by 16. For all remaining cases, the level 2 BLAS based DOT algorithm is superior to the others.

It can also be noticed that the memory related deficiency of DOT affects the performance of the level 3 BLAS codes. Our experiments with the unblocked GAXPY version of block decomposition inside the level 3 BLAS codes show that the best performance rises to about 274 Mflops for $N = 800$ and to about 292 Mflops for $N = 1600$.

The performance of the level 3 BLAS codes is only marginally better than the performance of level 2 BLAS. The overall performance for matrices of sizes bigger than 500 reaches 87–92% of the theoretical peak performance (equal to 333 Mflops). If we assume that the assembly language coded matrix multiplication (level 3 BLAS routine SGEMM) establishes the practical peak performance (approximately 312 Mflops [16]), the Gaussian elimination reaches 91–99% of this realistic peak.

In addition, the performance of all three versions of the level 3 BLAS codes is almost identical (the difference is not bigger than 1 Mflop) for the best blocksizes. Such blocksizes varied for different versions of the level 3 BLAS based codes and for different matrix sizes.

This leads us to the question what the optimal blocksize is. There have been some attempts by researchers from the LAPACK project [3, 14] to provide a theoretical basis for an automatic blocksize selection. For the time being, however, only the method of trial and error is available. Figure 6 presents the effect of the change in blocksize on the performance of the blocked algorithm for the matrix of size 1951. We obtained similar results for a variety of matrix sizes. Figure 6 also compares the performance of the level 3 BLAS codes with SGEMM and SGEMMS (Strassen's algorithm) in the update step. In order to compare the performance of codes using SGEMM and SGEMMS properly, the time (not the megaflop rate) was measured. For the matrix size $N = 1951$ the best result obtained using level 2 BLAS (DOT version) is 16.1 seconds.

Our results confirm the well known fact that once the blocksize is large enough, a change in the blocksize does not generate substantial changes in the effectiveness of the code. It is easy to see that when the blocksize is sufficiently large, the codes using SGEMMS become much more efficient than those using SGEMM. For an appropriately large blocksize, all three versions

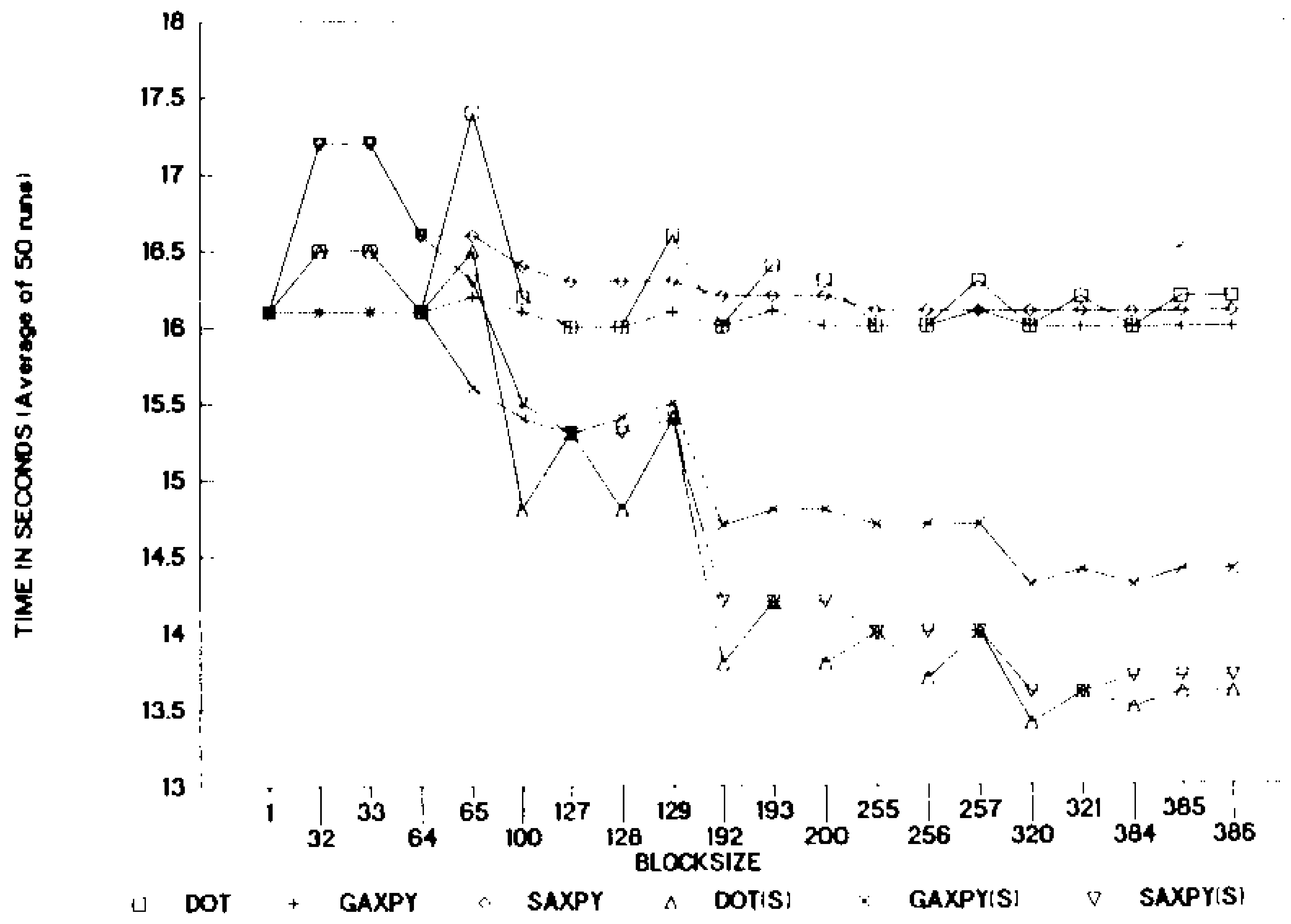

FIG. 6. Blocksize effect for $N = 1951$; results in seconds; "(S)" marks codes using Strassen's update.

using SGEMM behave in a similar way. (The performance of DOT and GAXPY is very close, and both slightly outperform the SAXPY version.) In the case of codes using Strassen's algorithm, DOT(S) and SAXPY(S) outperform GAXPY(S) in a whole range of large blocksizes.

One more point is worth mentioning. The best overall performance was achieved for blocksize 320. This situation was also observed for other large matrices. It can be explained by the fact that when the matrix size increases the amount of time spent in matrix updates increases in comparison with other operations [19]. Since the effects of Strassen's algorithm become more and more visible when the matrix size increases, we observe a tradeoff. Slight deficiencies in the decomposition step caused by the somewhat too large blocksize are overcome by the speed of Strassen's update. It can be predicted (see Table 2) that as the matrix size increases the optimal blocksize for the codes using Strassen's update will slowly increase. (For matrices of sizes 1950, 1951, and 1952—the largest matrices we experimented with—the optimal blocksize was 320.)

In Table 2 we compare the performance of the best versions of Gaussian elimination that do not use Strassen's algorithm (DOT or GAXPY inside the level 3 BLAS) with the blocked algorithm that does. For each matrix size, the version with the best performance is specified together with its blocksize. When the same result was obtained for different blocksizes, the smallest one is specified.

The results in Table 2 are in agreement with those presented in Figure 6 for large matrices. The DOT version is the best among the codes using Strassen's update, whereas the DOT and the GAXPY versions are clear winners for codes not using it. The optimal blocksize for codes using the SGEMM update is equal to 128. As predicted above, a slow increase in the optimal blocksize for codes using the SGEMMS update is observed.

The crossover point when using SGEMMS becomes significant is somewhere around $N = 300$. For practical purposes, however, it should be assumed that the effect of using SGEMMS in Gaussian elimination will become apparent for systems larger than $N = 400$.

TABLE 2
COMPARISON BETWEEN CODES USING SGEMMS and those not using it

	No Strassen			Strassen		
N	Time (sec)	Version[a]	Blocksize	Time (sec)	Version[a]	Blocksize
300	0.0681	D/G/S	128	0.0673	S	128
400	0.168	D/G/S	128	0.166	S	192
500	0.289	G	128	0.277	G	256
600	0.497	D/G	128	0.465	S	192
700	0.771	G	128	0.716	S	192
800	1.22	BL2		1.17	SG	320
900	1.63	G	192	1.47	S	256
1000	2.21	D/G	128	1.98	D	256
1100	2.94	D/G	128	2.63	D	192
1200	3.88	D	64	3.46	D	192
1300	4.80	G	128	4.24	D	192
1400	5.98	G	128	5.23	D	192
1500	7.34	G	128	6.33	D	256
1600	9.21	BL2		8.08	SG	320
1700	10.6	G	128	9.12	D	256
1800	12.7	D/G	128	10.7	S	320
1900	14.8	D/G	64	12.5	D	320

[a]BL2: level 2 BLAS code (GAXPY); D: DOT; G: GAXPY; S: SAXPY; SG: level 3 BLAS SAXPY with level 2 BLAS GAXPY.

4. CONCLUSIONS

We have shown that for the assembly coded versions of level 1, 2, and 3 BLAS routines provided by the Cray Research, Inc., it makes very little difference if the blocked or unblocked versions of the Gaussian elimination are used. It was also suggested that the Strassen's algorithm can be successfully used in the update step to increase the overall performance of the code. For large matrices, the gain in speed caused by SGEMMS is about 15%. As the matrix size becomes larger the impact of SGEMMS is expected to increase.

The author wishes to express his gratitude to Cliff Cyphers, who implemented a large part of the experiments on the Cray.

REFERENCES

1 R. Allen, S. Pruess, and L. F. Shampine, *Fundamentals of Numerical Computing*, Lecture notes—revised ed., Southern Methodist Univ., 1987.

2 C. H. Bischof, Fundamental Linear Algebra Computations on High-Performance Computers, Technical Report MCS-P150-0490, Argonne National Lab., 1990.

3 C. H. Bischof and P. G. Lacroute, An Adaptive Blocking Strategy for Matrix Factorization, Technical Report MCS-P151-0490, Argonne National Lab., 1990.

4 Cray Research, Inc., Math and Scientific Reference Manual, SR-2081 5.0.

5 M. J. Dayde and I. S. Duff, Level 3 BLAS in *LU* Factorization on Cray-2, ETA-10P and IBM 3090-200/VF, *Internat. J. Supercomput. Appl.* 3(2):40–70 (1989).

6 J. J. Dongarra and S. C. Eisenstat, Squeezing the most out of an algorithm in CRAY FORTRAN, *ACM Trans. Math. Software* 10(3):219–230 (1984).

7 J. J. Dongarra, Performance of Various Computers Using Standard Linear Equation Software, Technical Report CS-89-85, Univ. of Tennessee, 1990.

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

9 J. J. Dongarra, J. Du Croz, S. Hammarling, and R. J. Hanson, An extended set of FORTRAN basic linear algebra subprograms, *ACM Trans. Math. Software* 14(1):1–17 (1988).

10 J. J. Dongarra, F. G. Gustavson, and A. Karp, Implementing linear

algebra algorithms for dense matrices on a vector pipeline machine, *SIAM Rev.* 26:91–112 (1984).

11 K. Gallivan, W. Jalby, U. Meier, and A. Sameh, Impact of hierarchical memory systems on linear algebra algorithm design, *Internat. J. Supercomput. Appl.* 2(1):12–46 (1988).

12 K. Gallivan, J. R. Plemmons, and H. A. Sameh, Parallel algorithms for dense linear algebra computations, *SIAM Rev.* 32(1):54–135 (1990).

13 N. J. Higham, Exploiting Fast Matrix Multiplication within the Level 3 BLAS, Technical Report TR 89-984, Cornell Univ., 1989.

14 LAPACK Project Technical Reports, available from `drake@cs.utk.edu`.

15 C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh, Basic linear algebra subprograms for FORTRAN usage, *ACM Trans. Math. Software* 5(3):306–323 (1979).

16 M. Paprzycki and C. Cyphers, Multiplying matrices on the Cray—practical considerations, *CHPC Newsletter* 6(6):77–82 (1991).

17 M. Pernice, The Performance of *LU* Factorization Algorithms on the IBM 3090/600S, Technical Report USI-1, Utah Supercomputing Inst., 1990.

18 V. Strassen, Gaussian elimination is not optimal, *Numer. Math.* 13:354–356 (1969).

19 R. A. van de Geijn, LINPACK Benchmark on the Intel Touchstone GAMMA and DELTA Machines, Preliminary Report, Univ. of Texas, 1991.

Received 30 August 1991; final manuscript accepted 2 January 1992