Parallel Gaussian Elimination Algorithms on a Cray Y-MP

Marcin Paprzycki
Department of Mathematics and Computer Science
University of Texas of the Permian Basin
Odessa, TX 79762-0001, USA
Phone: +915 552 2258, Fax: +915 552 2374
paprzycki_m@gusher.pb.utexas.edu

Keywords: performance, parallel Gaussian elimination, Strassen's algorithm

Edited by: Jerzy R. Nawrocki

Received: September 18, 1994 **Revised:** January 24, 1995 **Accepted:** February 27, 1995

Various implementations of Gaussian elimination of dense matrices on an 8-processor Cray Y-MP are discussed. It is shown that when the manufacturer provided BLAS kernels are used the difference in performance between the best blocked implementations and Cray's routine SGETRF is almost negligible. It is shown that for large matrices Strassen's matrix multiplication algorithm can lead to substantial performance gains.

1 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. We are interested in a parallel solution based on the use of *BLAS* [9, 10, 15] primitives and blocked algorithms [1, 3, 6, 8, 12]. Since the release of Unicos 6.0 Cray provides a set of assembly coded parallel *BLAS* kernels as well as some additional routines; one of them is *SGETRF* which performs parallel Gaussian elimination with row interchanges. There exist situations [19, 23] in which Gaussian elimination with column interchanges is necessary, so it becomes important to know how much worse is the performance of the "home made" codes in comparison to *SGETRF*. It was shown [2, 14, 16] that using Strassen's matrix multiplication algorithm in the update step of the Gaussian elimination on a one-processor Cray Y-MP can lead to substantial time savings. We shall presently address the possibility of using parallel Strassen's matrix multiplication algorithm in the update step of parallel Gaussian elimination.

2 Level 3 BLAS based algorithms – implementation details

We compared the performance of different versions of Gaussian elimination on an 8-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. We investigated the performance of *DOT*, *GAXPY* and *SAXPY* versions of Gaussian elimination as described in [6, 11, 16]. Each consists of three operations performed on submatrices (blocks) of A: (1) the solution of a linear system for the multiple right hand sides (level 3 *BLAS* routine *_TRSM*), (2) the block update (level 3 *BLAS* routine *_GEMM*), and (3) the decomposition of a block of columns. Since Cray provides assembly coded routines, *STRSM* and *SGEMM*, only the last operation needs to be implemented independently.

To decompose a block of columns each of the three unblocked versions of the column oriented elimination can be used. In [16] it was shown that for the one-processor Cray Y-MP, for the long blocks of columns, the unblocked *GAXPY* and *DOT* versions are the most efficient. At the same time for larger numbers of processors the unblocked $GAXPY$ performs poorly [20]. Our experiments suggest that for multiple processors and for blocks of columns shorter than 1500 the unblocked $SAXPY$ based routine is more efficient than the unblocked DOT whereas as the number of rows in the block increases the DOT routine seems to become more efficient. For that reason we have selected the $SAXPY$ based decomposer for the blocked codes. To confirm this choice we have performed some experiments with the implementations using the unblocked DOT version of the decomposition step. For matrices of sizes up to 2600 (which was the largest size we have experimented with), all blocked codes with the $SAXPY$ based decomposer have outperformed the codes with the DOT based decomposer. It should be added as the matrix size increased that the difference in performance tended to decrease. as the matrix size increases the percent of the time spent in the decomposition step relative to the time spent in other steps decreases [22]; thus for very large matrices, even if the DOT based decomposer would be slightly more efficient its effect would be negligible. It is only for smaller matrices that the efficiency of a decomposer really matters and in those situations $SAXPY$ is the fastest.

3 Numerical results

Since in [16, 20] it was shown that when the assembly coded $BLAS$ kernels are used the difference between the performance of the best versions of blocked and unblocked codes is almost negligible, in the first series of experiments we have investigated the parallel performance of the level 2 $BLAS$ based codes. Figure 1 summarizes the 8-processor performance.

The effect of the system bus being saturated with data communication can be clearly observed. The $SAXPY$ variant is competitive only for very small matrices, whereas the DOT version is the leader for medium size matrices. Interestingly, the $GAXPY$ variant, which is slower than the others for most of the matrix sizes, becomes the performance leader for very large matrices. This gain, however, is only relative to the unsatisfactory performance of the other variants. A severe effect of memory bank conflicts can be observed

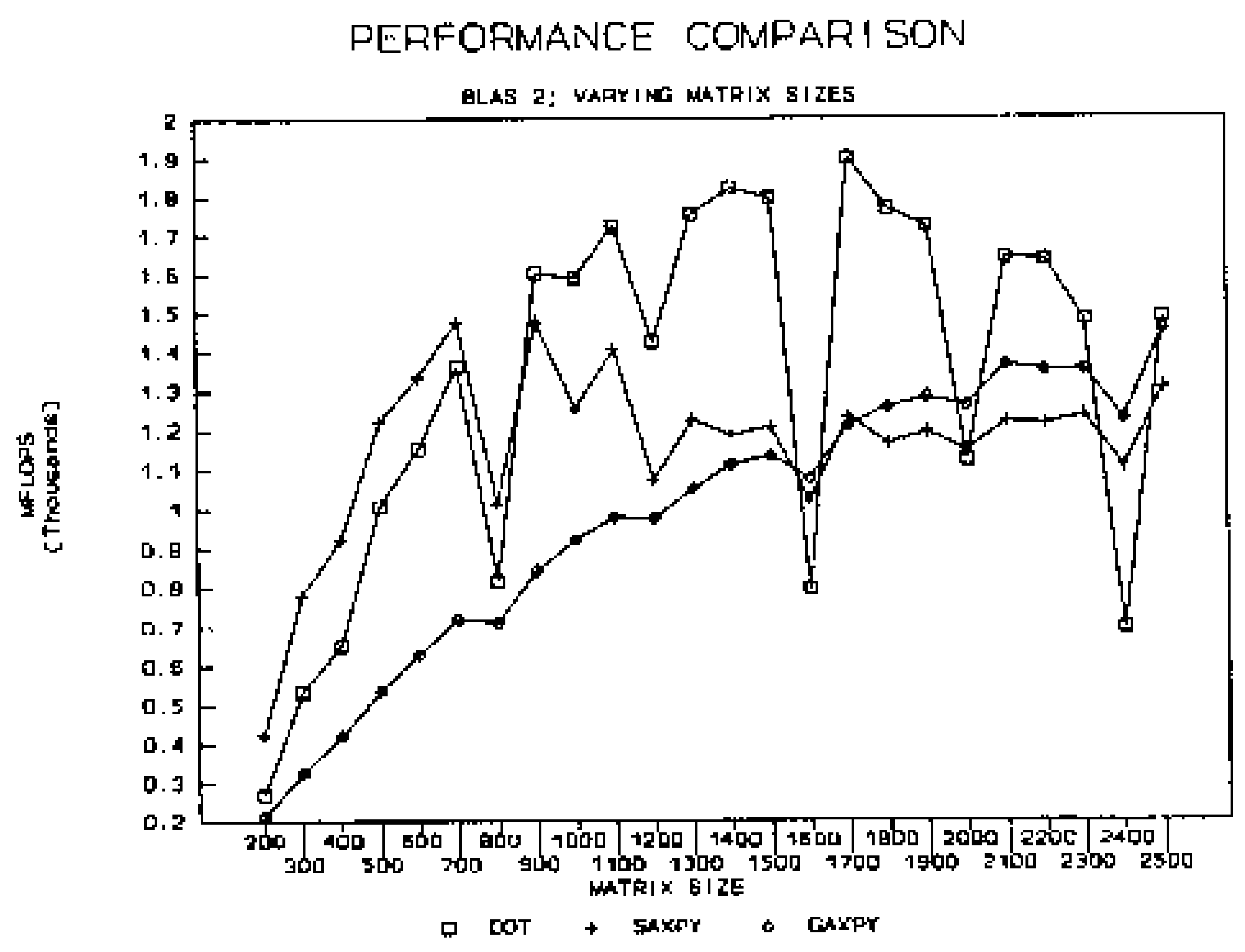

Figure 1: Comparison between the level 2 $BLAS$ based codes; 8 processors; results in MFlops.

for matrices of size 800, 1600 and 2400.

The second series of experiments was designed to investigate the effect of change in the blocksize on the performance of the blocked algorithms. There are some attempts by researchers from the $LAPACK$ project [4] to provide a theoretical basis for an automatic blocksize selection. For the time being, however, only the method of trial and error is available. Table 1 compares the performance of all three versions of the blocked algorithm for a variety of blocksizes when all 8 processors are used; the matrix size is 1600.

Blocksize	SAXPY	DOT	GAXPY
64	1.559	1.541	1.689
128	1.551	1.549	1.580
192	1.530	1.549	1.559
256	1.545	1.531	1.564
320	1.570	1.546	1.546
384	1.645	1.612	1.609

Table 1: Blocksize effect on the performance; 8 processors; time in seconds.

The best performance is obtained for blocksizes 192 and 256. In general, for large matrices the performance gain from blocking (over level 2 $BLAS$ based codes) was approximately 30%; the performance gain from using the best blocksize was additional 3-4%. These results conform to what was presented in [20] for smaller matrix sizes. At the same time our experiments indicate that (1) as the number of processors increases (for a fixed matrix size) the optimal blocksize increases and (2) as the matrix size increases (for a fixed number of processors) the optimal blocksize decreases. These results are illustrated in Tables 2 and 3.

This may be explained by the fact that as the number of processors increases greater amounts of data need to be transferred to keep them busy, and so a larger blocksize is required. As the matrix size increases a smaller number of columns needs to be transferred to provide the fixed number of processors with the same amount of data to work with. This effect is, of course, mediated by the communication capacity of the system bus. Thus, contrary to what is common in practice, for blocked algorithms performed on parallel computers, there seems to exists an *optimal blocksize per processor* (rather than a *fixed* optimal blocksize). Unfortunately the optimal blocksize per processor will vary from architecture to architecture and needs to be established experimentally.

The final set of experiments was directed at the possibility of using Strassen's matrix multiplication algorithm in the update step of parallel Gaussian elimination. In 1969, Strassen [21] showed that it is possible to multiply two matrices of sizes $N \times N$ in 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 certain trade-offs. The first one is between a gain in speed and an increase in required storage. In Cray's implementation (routine $\_GEMMS$), the additional work array of size $2.34 * N^2$ is required [5]. The second is between the depth of recursion and parallelization. The deeper the level of recursion the greater is the one-processor performance gain. At the same time, however, in Cray's implementation where recursion is applied only up to block of size of approximately 64 (and parallel $SGEMM$ is used to calculate the result), the effects of recursion are reduced [17].

There is one further important consideration concerning the stability properties of Strassen's algorithm since they are less favorable than those of the conventional matrix multiplication algorithm [7, 14]. In [14] Higham showed that for square matrices (where $N = 2^k$), if $\hat{C}$ is the computed approximation to C ($\hat{C} = AB + \Delta C$), then

$$\|\Delta C\| \leq c(N,N,N)u\|A\|\|B\| + O(u^2)$$

where u is a unit roundoff, and

$$c(N,N,N) = (\tfrac{N}{2^k})^{log_2 12}((2^k)^2 + 5 * 2^k) - 5N.$$

(For non-square matrices the function c becomes more complicated, but the overall result remains the same.) Observe that the error bound for standard matrix multiplication is $Nu\|A\|\|B\|+O(u^2)$. Therefore, the expected error growth should not be too serious in computational practice. This conclusion is also backed up by the results of our experiments.

Figure 2 presents the results of our experiments with Strassen's matrix multiplication in the update step of the Gaussian elimination for matrix sizes N = 600, ..., 2600. Following [2], the results are presented in calculated MFlops. (This allows for a clearer expression of the performance gains of the new algorithm even though Strassen's algorithm uses fewer operations.) We have decided to present only the $SAXPY$ and DOT based implementations since when multiple processors were used (for the Strassen as well as non-Strassen based implementations) the $GAXPY$ variant was much slower than the other two for all matrix sizes (this is primarily caused by the fact that the matrices involved in the update step are long and "thin"; see also [20]). We also present the performance of the Cray provided routine $SGETRF$ and the practical peak performance. For both Strassen as well as non-Strassen implementations the blocksize 265 was used.

A number of observations can be made. There is no additional gain from using the Cray provided routine $SGETRF$. This is very important whenever Gaussian elimination with column interchanges needs to be employed (e.g. in algorithms using alternate row and column elimination strategy [19, 23]). In [17], it was argued that the practical peak performance for the 8-processor Cray

Matrix size	200	600	1000	1400	1800	2200	2600
Optimal blocksize	512	320	256	320	256	256	192

Table 2: Optimal blocksize for a fixed number of processors (8) and increasing matrix size.

Number of Processors	1	2	3	4	5	6	7	8
Optimal blocksize	64	256	256	256	384	256	384	256

Table 3: Optimal blocksize for a fixed matrix size (2500) and increasing number of processors.

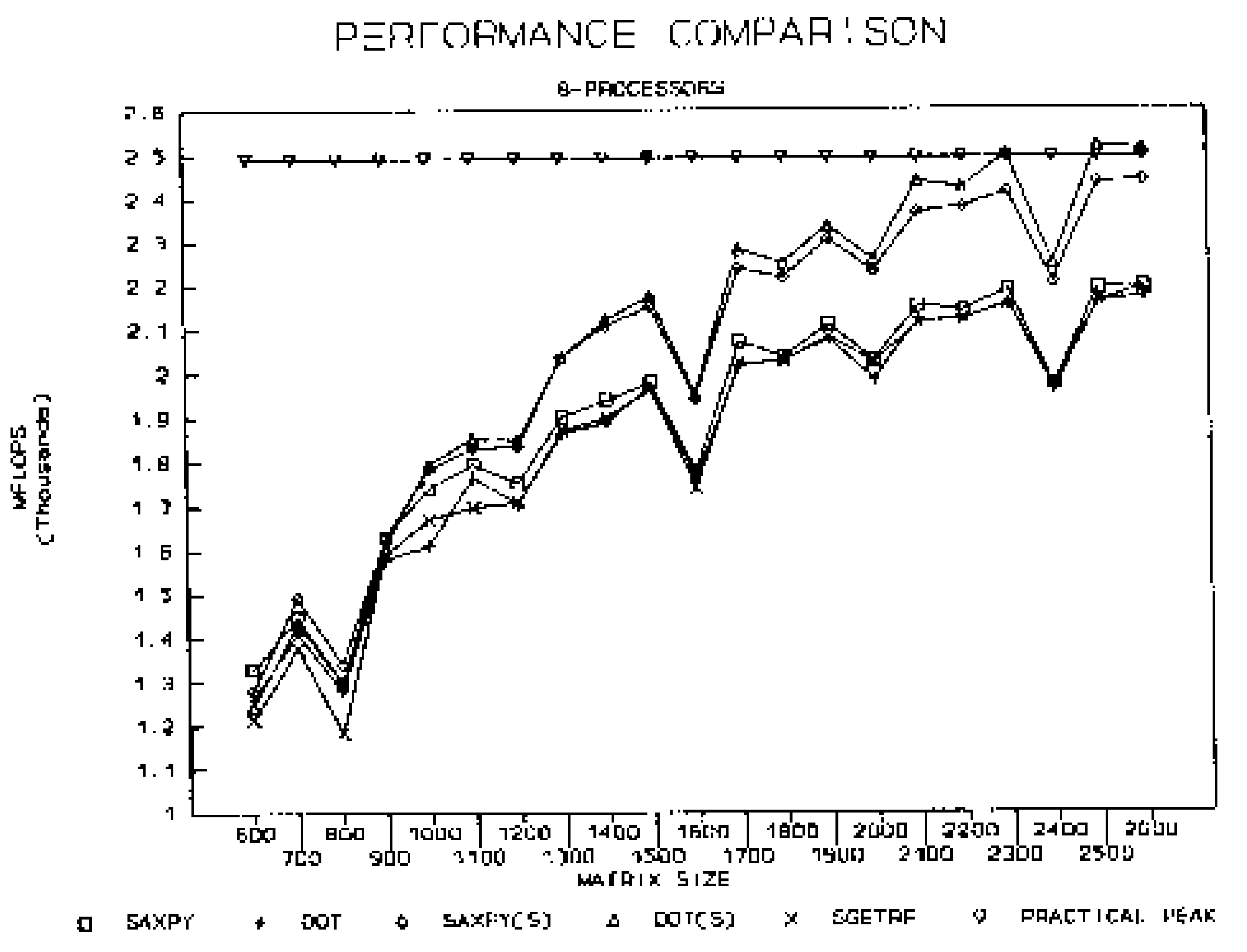

Figure 2: Performance comparison between Strassen and non-Strassen-based codes; 8 processors; results in MFlops; '(S)' marks codes using Strassen's update.

Y-MP is approximately 2490 MFlops. Therefore the blocked (non-Strassen) Gaussian elimination algorithms reach approximately 88% of this practical peak for large matrices. The time reduction obtained due to Strassen's matrix multiplication used in the update step of Gaussian elimination increases with the matrix size and reaches 13% for matrices of size 2600. Using Strassen's matrix multiplication is advantageous only for matrices of sizes larger than 1000. Approximate minimal matrix sizes for which the Strassen-based Gaussian elimination outperforms the non-Strassen versions for 1 – 8 processors are summarized in Table 4.

The results in Table 4 indicate clearly that the cross-over point migrates toward larger matrix sizes as the number of processors increases. Finally, the effect of memory bank conflicts is much smaller for the blocked codes than it is for the unblocked ones. This can be attributed to a more efficient data transmission pattern.

4 Conclusions

In the paper we have studied the performance characteristics of the *BLAS* based blocked Gaussian elimination for large dense matrices on an 8-processor Cray Y-MP. We have shown that the "home made" implementations of *SAXPY* and *DOT* versions of Gaussian elimination perform very closely to the Cray provided routine *SGETRF*. We have suggested that the standard notion of blocksize used for single processor blocked algorithms should be replaced by the notion of blocksize per processor for shared memory multiprocessor systems. We have also shown that if stability is not a serious consideration then, for large matrices, using parallel Strassen's matrix multiplication algorithm in the update step leads to substantial time savings.

5 Acknowledgement

The computer time grant from Cray Research Inc., is kindly acknowledged. The author wishes to express his gratitude to Chris Hempel (Cray Research Inc., Austin) for his help in running the experiments as well as in obtaining the computer time grant, anonymous referees for helping to prepare a better paper and to Katarzyna Paprzycka for the help with English.

References

[1] 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

Number of Processors	1	2	3	4	5	6	7	8
Cross-over point	400	500	600	700	800	900	1000	1100

Table 4: Approximate cross-over point between the Strassen-based and non-Strassen Gaussian elimination for increasing number of processors.

[2] Bailey, D.H., Lee, K., Simon, H.D., Using Strassen's Algorithm to Accelerate the Solution of Linear Systems, *The Journal of Supercomputing*, 4, 1990, 357-371

[3] Bischof, C.H., Fundamental Linear Algebra Computations on High-Performance Computers, Technical Report MCS-P150-0490, Argonne National Laboratory, 1990

[4] Bischof, C.H., Lacroute, P.G., An Adaptive Blocking Strategy for Matrix Factorization, Technical Report MCS- P151-0490, Argonne National Laboratory, 1990

[5] Cray Research, Inc., Math and Scientific Reference Manual, SR-2081 5.0.

[6] 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

[7] Demmel, J.W., Higham, N.J., Stability of Block Algorithms with Fast Level 3 BLAS, Numerical Analysis Report No. 188, University of Manchester, 1991

[8] Demmel, J.W., Higham, N.J., Schreiber, R.S., Block LU Factorization, Numerical Analysis Report No. 207, University of Manchester, 1992

[9] 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

[10] 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

[11] 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

[12] 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

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

[14] Higham, N.J., Exploiting Fast Matrix Multiplication Within the Level 3 BLAS, *ACM Transactions on Mathematical Software*, 16 (4), 1990, 352-368

[15] 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

[16] Paprzycki, M., Comparison of Gaussian Elimination Algorithms on a Cray Y-MP, *Linear Algebra and its Applications*, 172, 1992, 57-69

[17] Paprzycki, M., Parallel Matrix Multiplication – Can We Learn Anything New?, *CHPC Newsletter*, 7 (4), 1992, 55-59

[18] Paprzycki, M., Cyphers, C., Multiplying Matrices on the Cray – Practical Considerations, *CHPC Newsletter*, 6 (6), 1991, 77-82

[19] Paprzycki, M., Gladwell, I., Using Level 3 BLAS to Solve Almost Block Diagonal Systems, in: Dongarra, J., Kennedy, K., Messina, P., Sorensen, D.C., Voigt, R.V., (eds.), *Proceedings of The Fifth SIAM Conference on Parallel Processing for Scientific Computing*, SIAM, Philadelphia, 1992, 52-62

[20] Sheikh, Q., Performance of Block Matrix Factorization Algorithms and LAPACK on CRAY Y-MP and CRAY 2, in: Dongarra, J., Messina, P., Sorensen, D.C., Voigt, R.G., (eds.) *Parallel Processing for Scientific Computing*, SIAM, Philadelphia, 1989

[21] Strassen, V., Gaussian Elimination is not Optimal, *Numerical Mathematics*, 13, 1969, 354-356

[22] van de Geijn, R.A., LINPACK Benchmark on the Intel Touchstone GAMMA and DELTA Machines, Preliminary Report, University of Texas, 1991

[23] Varah, J.M., Alternate Row and Column Elimination for Solving Certain Linear Systems, *SIAM Journal on Numerical Analysis*, 13, 1976, 71-75