# Blocked LU Factorization in Engineering Applications on a Minisupercomputer

*A. Gaber Mohamed*
*Geoffrey Fox*

**CRPC-TR91131**
**April 1991**

Center for Research on Parallel Computation
Rice University
P.O. Box 1892
Houston, TX 77251-1892

SCCS - 94
CRPC-TR91131

# "Note: Blocked LU Factorization in Engineering Applications on a Minisupercomputer"

A. Gabor Mohamed and Geoffrey C. Fox

Sumbitted to:

*Microcomputers in Civil Engineering,*
An International Journal
Elsevier Applied Science

for:

ASCE 1991 Engineering Mechanics Conference
Mechanics Computing in the 1990's and Beyond

April 1991

Syracuse Center for Computational Science
Syracuse University
111 College Place
Syracuse, New York 13244-4100
<sccs@npac.syr.edu>
(315) 443-1723

# Note
# Blocked *LU* Factorization in Engineering Applications on a Minisupercomputer

A. Gaber Mohamed [1] and Geoffrey C. Fox [1]

## 1 Abstract

This note discusses methods of implementing the Level 3 BLAS in LU factorization used in engineering applications on the Alliant FX/80 minisupercomputer. Three ways of expressing the LU factorization in terms of blocked algorithms using Level 3 BLAS are considered. We also compare the performance of the parallelism within the computational kernels using a noblock algorithm that employs Level 1 and Level 2 BLAS with that obtained over the kernels when using blocked LU with the Level 3 BLAS.

## 2 Introduction

The importance of parabolic, elliptic partial differential equations and ordinary differential equations in engineering problems is well known. Most physical phenomena are modeled either by a system of PDEs or ODEs. Using a discretization technique like finite differences or finite elements, we end up with a system of linear algebraic equations. Even in nonlinear phenomena, one might solve a nonlinear system by iterating over the solution of a sequence of linear systems [1,2,3,4].

Consider the solution of the dense system of linear equations,

$$Ax = b, \tag{1}$$

where $A$ is an $n$-by-$n$ matrix and $b$ is a vector of dimension $n$.

[1]Syracuse Center for Computational Science, Syracuse University, Syracuse, NY 13244

One method of solving this problem is to proceed by first factorizing $A$ into lower and upper triangular matrices $L$ and $U$, *i.e.*,

$$A = LU, \tag{2}$$

then solving for $y$ and $x$ in substitution steps:

$$Ly = b \quad and \quad Ux = y. \tag{3}$$

In programs for applications of this type, more than 50% of CPU time is usually spent in matrix factorization. This occurs because most standard programming practices in Fortran result in more memory accesses than floating point operations. Our previous numerical experiments showed that traditional linear algebra-type codes do not achieve high performance on shared-memory multiprocessors because of lack of data locality [5]. Data locality is the fundamental problem in parallel computing and has great influence on the performance of such machines. Use of block-based algorithms is one of the most efficient ways to improve the performance of shared memory machines.

Dongarra, Gustavson, and Karp [6] discussed six ways of implementing the $LU$ factorization obtained by ordering the three nested loops that constitute the algorithm. The following generic Gaussian elimination algorithm explains the nomenclature:

```
do -------------
  do -------------
    do -------------
      a(i,j) = a(i,j) - a(i,k)*a(k,j)/a(k,K)
    end do
  end do
end do
```

Since Fortran is column-oriented, only three of the six forms called $JIK$-SDOT (also known as Crout's algorithm), $JKI$-GAXPY, and $KJI$-SAXPY, are suitable for Fortran applications. LAPACK [7,8] is a portable public linear algebra library based on the use of parallelized BLAS kernels, supplied by the vendors of different global shared-memory machines. The LAPACK factorization module is called DGETRF and uses the $JIK$-SDOT algorithm

if the block size $NB$ is $> 1$, and the parallelized Level 3 BLAS kernels– GEMM for multiplying two matrices, and TRSM for solving a set of triangular systems. If the block size is one, it uses an unblocked factorization with the parallelized Level 1 BLAS and Level 2 BLAS kernels– GEMV for multiplying a matrix by a vector, and TRSV for solving a triangular system. The amount of arithmetic is exactly the same; however, the data access and updating patterns are quite different.

Our early testing of a preliminary version of LAPACK showed that performance of the noblock algorithm (DGETF2) is superior to the blocked algorithm coded in DGETRF. This motivated us to study the implementation of the $JIK$-SDOT from LAPACK on the Alliant FX/80 and to develop different blocked algorithms like $JKI$-GAXP and $KJI$-SAXPY (a noblock version of it has been used for years in LINPACK). Here, we compare their performance with LAPACK's block and noblock routines.

# 3 Impact of the Alliant FX/80 Architecture

The FX/80 is a shared memory parallel computer with six interactive processors (IPs) and eight pipelined advanced computational elements (ACEs). Each ACE contains eight 64-bit floating-point registers 32 elements long. A concurrency control bus connects the eight ACEs and acts as a synchronization facility. The ACEs share a 512k byte write-back cache. The cache is connected to memory by a memory bus [9].

All the arithmetic computations are performed at the top of this hierarchy (in vector registers). Therefore, the key to efficiency is to keep active data as close as possible to the top of the hierarchy.

The Alliant Scientific Library offers assembly-coded computational kernels for the basic operations in linear algebra. These kernels are known as the BLAS (Basic Linear Algebra Subprograms). Different levels of BLAS are available. For example, if the vectors and matrices involved are of order $N$, the Level 1 BLAS provides vector computations of order $O(N)$, the Level 2 BLAS provides matrix-vector computations of order $O(N^2)$, and Level 3 BLAS provides matrix-matrix computations with $O(N^3)$ operations. Therefore, the Level 1 BLAS and Level 2 BLAS do not possess as good a ratio of operations to data movement as the Level 3 BLAS to achieve high performance when exploiting concurrency and vectorization.

# 4 Block Factorization using BLAS Kernels

Of the six ways of implementing LU factorization, using partial pivoting with row interchanges, that were discussed by Dongarra, Gustavson, and Karp [6], we describe the three column-oriented variants using their nomenclature. We consider the block implementation of the three algorithms.

In all cases work is done on blocks with $NB$ columns using a matrix-vector based elimination scheme to reduce each block column in turn. Thus, we consider the three block-column variants and, in each case, all pivoting is performed only within a noblock algorithm. Any permutation resulting from this pivoting must be applied to the remainder of the matrix. The noblock algorithms $JKI$-noblock and $JIK$-noblock are considered. A $JKI$-noblock is developed based on a pseudo-code described by Dayde and Duff [10]. The $JIK$-noblock or DGETF2 from LAPACK [7,8] is used. Our numerical experiments show that the $JIK$-noblock's performance is superior to the $JKI$-noblock's performance. Therefore, in all cases, we use the $JIK$-noblock or DGETRF subroutine from LAPACK.

## 4.1 Block $JIK$-SDOT (Crout's Algorithm)

In the $JIK$-SDOT algorithm, at the $k$th step of the elimination process, one block column of $L$ and one block row of $U$ are computed. These computations require the following operations:

- updating of the diagonal and subdiagonal blocks of the $k$-th block column (GEMM);
- factorizing the $k$th block column into LU factors, performing numerical pivoting and using Level 2 BLAS (GEMV and TRSV);
- updating the $k$th block row of U (GEMM); and
- computing the $k$-th block row of U (TRSM).

## 4.2 Block $JKI$-GAXPY

In the $JKI$-GAXPY algorithm, at the $k$th step of the elimination, a block column of both matrices $L$ and $U$ is computed. These computations require

the following operations:

- computing the $k$th superdiagonal block of U (TRSM);
- updating the $k$th diagonal and subdiagonal blocks (GEMM); and
- factorizing the $k$th block column into LU factors, performing numerical pivoting and using Level 2 BLAS (GEMV and TRSV).

### 4.3 Block *KJI*-SAXPY

At the $k$th step of the elimination a block column of L and a block row of U are computed and the corresponding transformations are applied to the remaining reduced matrix. This algorithm requires the following operations:

- factorizing the $k$th block column into LU factors, performing numerical pivoting and using Level 2 BLAS (GEMV and TRSV);
- computing the $k$th block row of U (TRSM); and
- updating the remaining matrix using a block outer product (GEMM).

To obtain a copy of all the software used in this study, send a one-line e-mail message "send index" to allus@netlib.npac.syr.edu. Allus is a free software distribution electronic service. The index lists information on how to access all the programs used in this study. Users who have problems accessing these programs should send e-mail to the authors at agm@nova.npac.syr.edu.

## 5 Numerical Results

In all cases, the assembly-coded BLAS routines Level 1, Level 2, and Level 3 from the Alliant scientific library are used, as well as some routines from LAPACK. The specific BLAS routines used are DGEMM, DTRSM, DGEMV, DSCALL, IDAMAX, and DSWAP. LAPACK routines include: JIK-noblock (DGETF2) for partial pivoting with row interchanges; DLASWP to perform a series of row interchanges on blocks of the matrix $A$ using the DSWAP routine from the Alliant library; and DGETRS for solving the system $Ax = b$

after the matrix $A$ is $LU$ factorized by any of the three block-$LU$ factorization routines considered in this study. Also, from the LAPACK library, the routines XERBLA for error handling and the LSAME auxiliary function are used. The performance is measured in MFLOPS (Million Floating-point Operations Per Second) for solving the whole system $Ax = b$, *i.e.*, factorization by any of the three blocked algorithms and solution by DGETRS. The performance is measured on a stand-alone basis for a matrix of size $N = 100, 200, ....., 1000$ on 8 ACEs. The compiler options and flags used in all cases are "-O -DAS -lmath."

Our numerical experiments of Crout's method from LAPACK, after necessary modifications to use the FX/BLAS, show that the performance for $NB = 1$ is higher than that for $NB = 32$ if the matrix size is $N \leq 500$, as shown in Figure 1. The code is written to use the noblock algorithm if $NB = 1$ and the blocked version if $NB > 1$. The noblock algorithm is built around the use of Level 2 BLAS (GEMV and TRSV). Level 2 BLAS does not possess a good enough computational intensity (ratio of operations to data movement) to achieve high performance when exploiting concurrency and vectorization. However, parallelization at this level can provide a reasonable performance improvement when efficient parallelization tools for fine granularity (especially low-cost synchronizations) are available. The hardware-controlled microtasking provided by the Alliant is a good example of this. It is the hardware concurrency control in the Alliant FX/80 that explains why the assembly-coded Level 2 BLAS can provide a reasonable performance on this machine.

On the other hand, Dayde and Duff [10] reported that their numerical experiments, on the CRAY-2, Cyber 205, and IBM 3090-200/VF showed that the $JIK$-SDOT version was uniformly the worst. This is because much of the updating is done on a block row of U where the vector lengths are the same as that of the block size. Our results also show that $JIK$-SDOT is penalized by the short vector length inherent in this algorithm. This is confirmed by our results shown in Figure 1. It is clear from Figure 1 that the performance improves as the block size increases. The best performance is obtained for the block size equivalent to the total number of computational elements in the complex multiplied by 32 (256 in our case) since the number of ACEs used is 8.

Figures 2-5 show the performance comparison for the three column-oriented algorithms $JIK$-SDOT, $JKI$-GAXPY, and $KJI$-SAXPY for a block size of

32, 64, 128, and 256 on a complex size of 8 and stand-alone timing with the performance of the $JIK$-noblock algorithm ($NB = 1$ and using Level 1 and 2 BLAS from the Alliant library). $KJI$-SAXPY achieves the best performance for all block sizes. The highest performance is obtained for a block size of 64, and this is in agreement with the experimental results reported by Gallivan, Jalby, Meier, and Sameh [11]. Although Dongarra, Gustavson, and Karp [6] stressed the different access patterns of these three algorithms; for example, $KJI$-SAXPY requires about twice as many transfers to memory as $JIK$-SDOT and $JKI$-GAXPY, we do not see the effect of this in our results. This is because memory and cache management mechanisms mask such differences.

# 6 Conclusion

We have described the Fortran-oriented methods for block $LU$ factorization on a shared memory parallel vector minisupercomputer. These methods are also ready for portable implementations on other shared memory parallel vector computers. Our numerical experiments and performance comparisons showed the following:

- The block $JIK$-SDOT algorithm is very poor when block size is small due to the fact that most vector lengths in this algorithm are the same size as the block.
- The block $KJI$-SAXPY algorithm's performance is superior to all other blocked algorithms for any block size.
- The noblock $JIK$-SDOT algorithm's performance is superior to all blocked algorithms for a matrix of size $N \leq 500$. This is due to the built-in hardware concurrency control in the Alliant FX/80.

We recommend blocked $LU$ factorization parallelism over the assembly-coded Level 3 BLAS for sufficiently large problems. Efficient utilization of the hardware and assembly-coded BLAS Level 1 and Level 2 should be used for small problems.

# 7 References


1. Mohamed, A.G., and Valentine, D.T., "Taylor's Vortex Array: A New Test Problem for Navier-Stokes Solution Procedures," A Chapter in *Solution of Superlarge Problems in Computational Mechanics*, edited by Kane, J.H., Carlson, A.D., and Cox, D. L., pp. 167-181, Plenum, New York, 1989.

2. Fox, G., Johnson, M., Lyzenga, G., Otto, S., Salmon, J., and Walker, D., *Solving Problems on Concurrent Processors*, Vol. I, Prentice Hall, New Jersey, 1988.

3. Mohamed, A.G., and Valentine, D.T., "Numerical Predictions of Turbulent Flow in an Annular Pipe," Proceedings of the ASME International Computers in Engineering, Vol. II, pp. 471-479, Boston, MA, August 1990.

4. Mohamed, A.G., Valentine, D.T., and Hessel, R.E., "Numerical Study of Laminar Separation Over an Annular Backstep," Accepted for publication and in press at *Computers & Fluids*, Jan. 1991.

5. Mohamed, A.G., "Block-based Solvers for Engineering Applications," *Mechanics Computing in the 1990's and Beyond*, Proceedings of the ASCE Engineering Mechanics Speciality Conference, Columbus, Ohio, May 19-22, 1991.

6. Dongarra, J., Gustavson, F.G., and Karp, A., "Implementing Linear Algebra Algorithms for Dense Matrices on a Vector Pipeline Machine," *SIAM Review*, Vol. 26, No. 1, pp. 91-112, Jan. 1984.

7. Anderson, E., and Dongarra, J., *LAPACK Working Note 18: Implementation Guide for LAPACK*, University of Tennessee, CS-90-101, April 1990.

8. Anderson, E., Dongarra, J., Greenbaum, A., McKenney, A., Du Croz, J., Hammarling, S., Demmel, J., Bischof, C., and Sorensen, D., " LAPACK: A Portable Linear Algebra Library for High-Performance Computers," *Supercomputing 90*, pp. 2-11, Dec. 1990.

9. Alliant Computer Systems Corporation, *FX/Series Architecture Manual* , Littleton, MA, 1988.

10. Dayde, M. J., and Duff, I. S., "Level 3 BLAS in LU Factorization on the CRAY-2, ETA-10P, and IBM 3090-200/VF," *The International Journal of Supercomputer Applications*, Vol. 3, No. 2, pp. 40-70, Summer 1989.

11. Gallivan, K., Jalby, W., Meier, U. and Sameh, A., "Impact of Hierarchical Memory Systems on Linear Algebra Algorithm Design," *The International Journal of Supercomputer Applications*, Volume 2, No.1, pp. 12-48, Spring 1988.

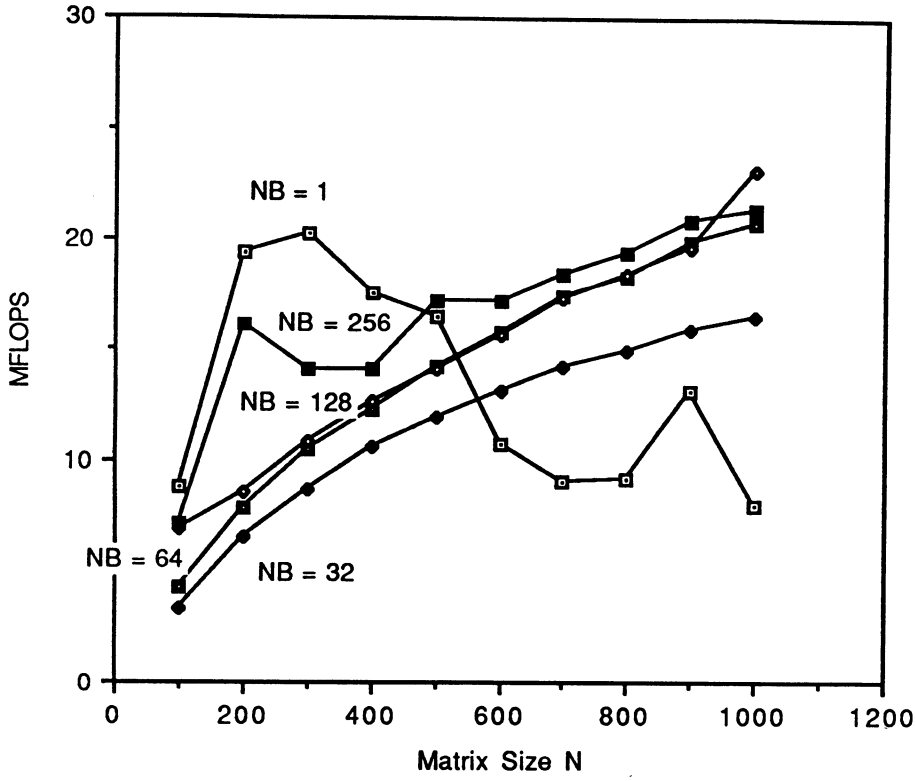


Figure 1 : Performance of blocked JIK-SDOT for different block size NB.

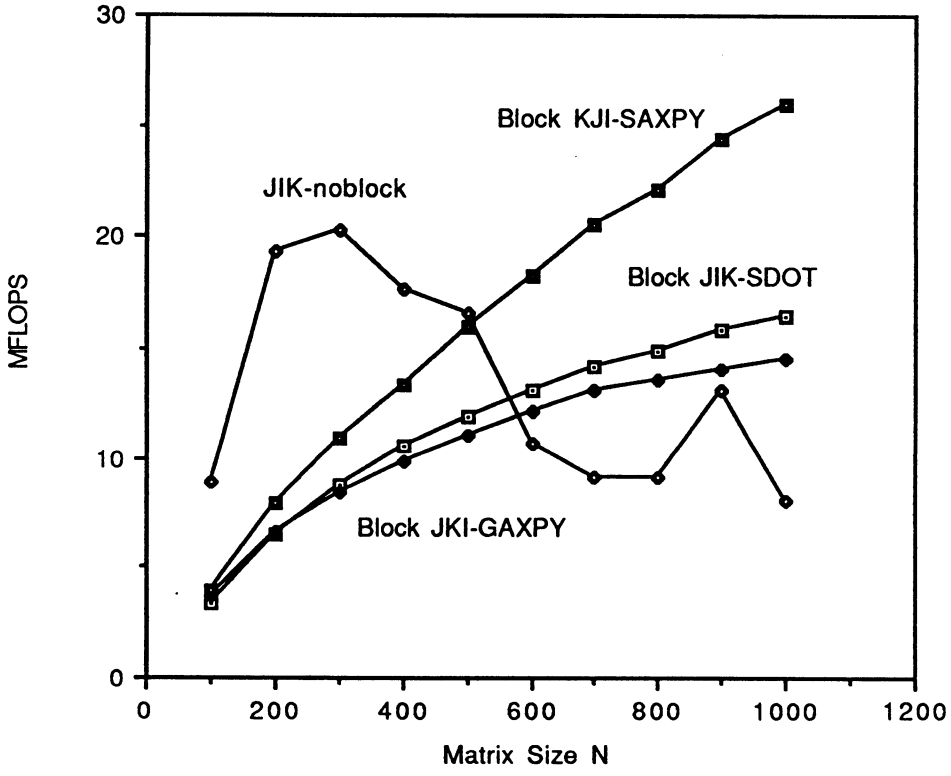


Figure 2 : Performance of different blocked algorithms for NB = 32 and JIK-noblock.

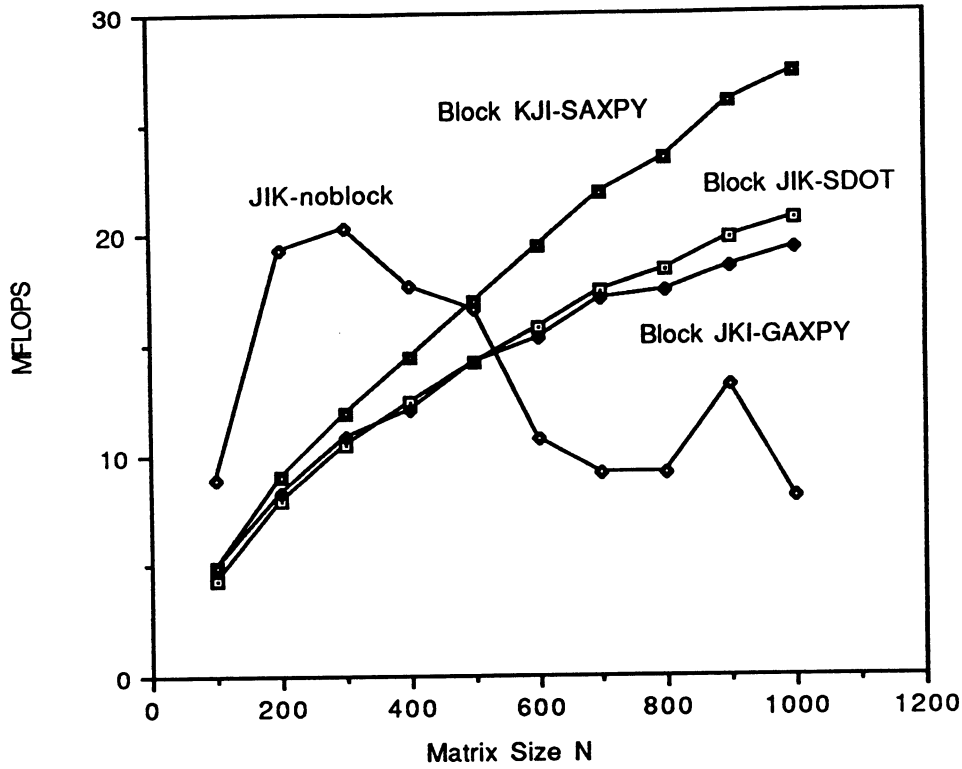


Figure 3 : Performance of different blocked algorithms for NB = 64 and JIK-noblock.

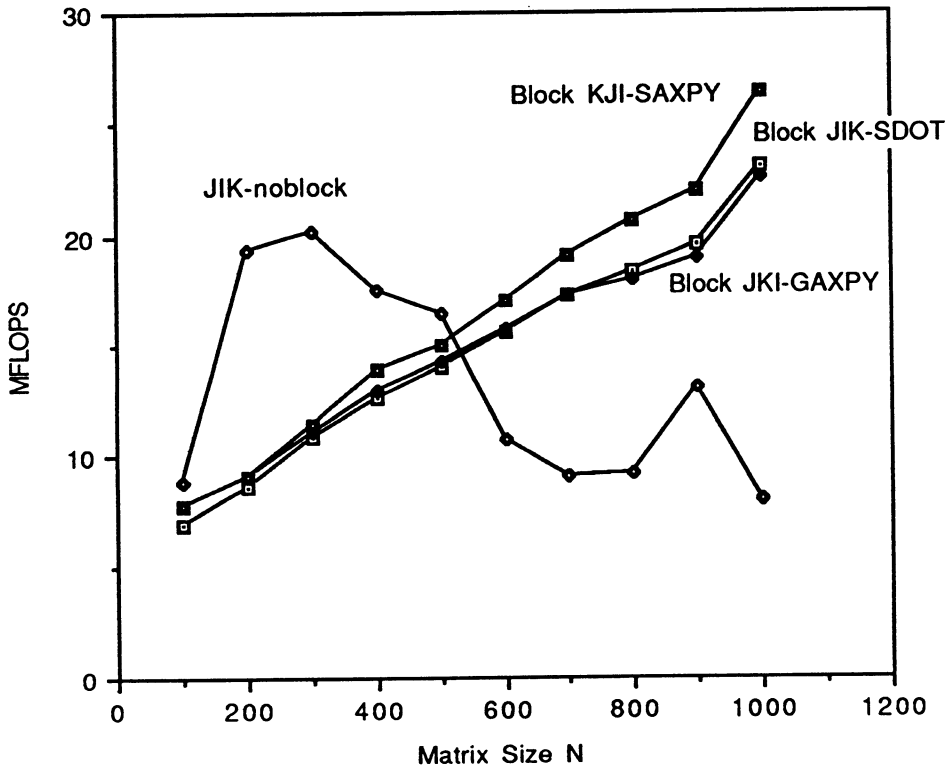


Figure 4 : Performance of different blocked algorithm for NB = 128 and JIK-noblock.

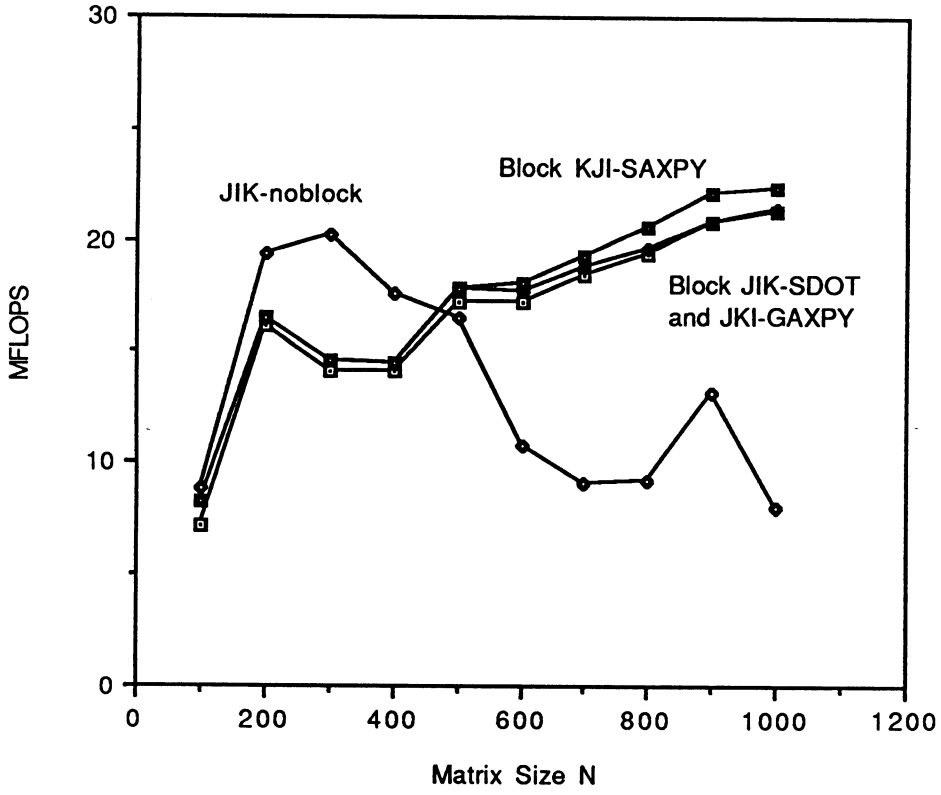


Figure 5 : Performance of different blocked algorithms for NB = 256 and JIK-noblock.