

## TIMELY COMMUNICATIONS

*Under the "timely communications" policy for the SIAM Journal of Scientific and Statistical Computing, papers that have significant timely content and do not exceed five pages automatically will be considered for a separate section of this journal with an accelerated reviewing process. It will be possible for the note to appear approximately six months after the date of acceptance.*

### THE USE OF BLAS3 IN LINEAR ALGEBRA ON A PARALLEL PROCESSOR WITH A HIERARCHICAL MEMORY\*

KYLE GALLIVAN†, WILLIAM JALBY†‡ AND ULRIKE MEIER†§

**Abstract.** This note describes work at CSRD which shows that a third level of the BLAS (BLAS3) is needed to achieve high-performance on multivector processors with a shared hierarchical memory.

**Key words.** BLAS3 or third-level BLAS, numerical linear algebra, numerical software, parallel computing, cache management

**AMS(MOS) subject classifications.** Primary 68B99; secondary 65F05, 65F25, 65F30

**1. Introduction.** The design of efficient numerical linear algebra codes which are also reasonably portable for vector machines such as the CRAY or CYBER 205 has been simplified considerably by the use of standard kernel subroutines known as the BLAS [3]. These routines are primitives based on vector-vector operations such as dotproduct and DAXPY ( $v = a + \gamma b$ ). The success of these routines depends upon the fact that many linear algebra algorithms are easily expressed in terms of vector-vector operations and that for high-performance pipelined architectures the optimization of performance is straightforward (the longer the vectors the better). Further consideration of algorithm performance has led to the development of the extended BLAS (called BLAS2 below) [4]. These routines are primitives based on matrix-vector operations. BLAS2 retains all of the positive aspects of the BLAS and possesses some unique advantages. On register-to-register vector machines the matrix-vector operations provide much more efficient register management and are able to more efficiently exploit memory bandwidth. Further, the presence of two-dimensional parallelism allows the BLAS2 primitives to be efficiently implemented on the newer supercomputers which provide concurrency as well as vectorization.

Unfortunately, most of the multivector processors make use of a hierarchical memory system, typically a small fast cache and a slower large main memory, to provide data at the required rates (ST-100, CRAY 2, ETA 10 and CEDAR). For these machines a third factor which influences algorithm performance is added to concurrency and vectorization. Algorithms must contain a reasonable amount of data locality in order to make full use of the resources of a multivector processor with a shared hierarchical memory. The data locality present in the BLAS and BLAS2 primitives and algorithms expressed in terms of these primitives is minimal. Hence, neither of

\* Received by the editors October 20, 1986; accepted for publication January 21, 1987. This work was supported in part by the National Science Foundation under grants DCR84-10110 and DCR85-09970, the U.S. Department of Energy under grant US DOE DE-FG02-85ER25001, and the IBM Donation.

† Center for Supercomputing Research and Development, University of Illinois, Urbana, Illinois 61801.

‡ This author was on leave from Institut National de Recherche en Informatique et en Automatique, domaine de Voluceau, Rocquencourt, B.P.105 78150, Le Chesnay, France.

§ This author was on leave from ZAM, KFA Juelich GmbH, Postfach 1913, D-5170 Juelich, West Germany.

the first two levels of the BLAS is a suitable basis for algorithm development on such architectures.

Recent work at CSRD has demonstrated that a third level of the BLAS is needed in order to design algorithms which efficiently exploit the resources of a multivector processor with a hierarchical shared memory system [1], [5], [6]. This third level of the BLAS (called BLAS3 below) comprises primitives based on matrix-matrix operations. These primitives possess as much exploitable concurrency and vectorization as BLAS and BLAS2 primitives, if not more, and their data locality is far better. The numerical library under development for the CEDAR machine uses "block" methods implemented via BLAS3 primitives for many of the basic linear algebra tasks such as matrix factorization.

The effect of data locality, concurrency and vectorization on the performance of an algorithm can be investigated by writing the total time required to perform the algorithm as

$$T = T_a + T_l = n_a \tau_a + n_l \tau_l$$

where  $T_a$ ,  $T_l$ ,  $n_a$  and  $n_l$  are the arithmetic time (the time spent computing assuming all operands are in cache), data load time (the time spent transferring data from main memory to cache), number of operations and number of data loads, respectively. The quantities  $\tau_a$  and  $\tau_l$  defined by this expression can be viewed as the "average" times for arithmetic and data load operations. The component  $T_a$  is largely the result of satisfying the aspects of concurrency and vectorization while data locality dictates the size of  $T_l$ . The effect of data locality on performance can be investigated by expressing the relative amount of time spent loading data in terms of a cache-miss ratio and a cost ratio which are  $\lambda = \tau_l / \tau_a$  and  $\mu = n_l / n_a$ , respectively. The relative amount of time spent loading data is then  $T_l / T = \mu / (\lambda^{-1} + \mu)$ . In [5] and [6] it is shown that by analyzing the data locality of a linear algebra algorithm via these two parameters the influence of the memory system can be reduced to a negligible amount for a large number of machines. The algorithm designer is then free to concentrate on the characteristics of his particular machine and their influence on the arithmetic time of the algorithm under consideration. For many algorithms this technique of optimizing the components  $T_a$  and  $T_l$  separately and then choosing parameters of the final algorithm from within the intersection of regions of near-optimal behavior for each of the terms yields remarkable results. This is the basic technique that has been used at CSRD in analyzing the performance of algorithms in the numerical library being developed for the CEDAR project.

This correspondence contains a brief survey of the results of some of the work done at CSRD which led to the recommendation of the use of the BLAS3 in the design of algorithms for the numerical library on the CEDAR machine and shared hierarchical memory machines in general. First, the results of the analysis of the basic BLAS3 primitive of  $C \leftarrow C + AB$  are discussed in § 2 and experimental results clearly demonstrating the superiority of BLAS3-based algorithms are presented in § 3.

**2. Analysis of a BLAS3 primitive.** The first task which results from recommending the use of the BLAS3 primitives in algorithm design is that of generating efficient algorithms for the primitives themselves. In this section the results of analyzing the BLAS3 primitive  $C \leftarrow C + AB$  presented in [5] are discussed. If  $A$  and  $B$  are vectors then this primitive is the rank-1 update of BLAS2. In BLAS3 usage, this primitive is usually a rank- $k$  update but it can, of course, be used as a dense matrix multiplication primitive when  $A$ ,  $B$  and  $C$  are all large. One of the most commonly used instances

of this primitive is the case where  $C$  is  $n_1$  by  $n_3$ ,  $A$  is  $n_1$  by  $n_2$  and  $B$  is  $n_2$  by  $n_3$ . The values of  $n_1$  and  $n_3$  are assumed large while  $n_2$  may or may not be. It is assumed that the multivector processor of interest contains  $p$  vector processors which share a small high-speed cache and a large slower main memory (for example an ALLIANT FX/8 which is used as a single CEDAR cluster).

Let  $A$ ,  $B$  and  $C$  be partitioned into submatrices of dimension  $m_1$  by  $m_2$ ,  $m_2$  by  $m_3$  and  $m_1$  by  $m_3$ , respectively. The goal of the analysis is to determine block sizes which deliver near-optimal performance of this BLAS3 primitive on a machine with the assumed architecture.

The algorithm for obtaining the matrix  $C$  is as follows:

```

do  $i = 1, k_1$ 
  do  $k = 1, k_2$ 
    do  $j = 1, k_3$ 
       $C_{ij} = C_{ij} + A_{ik} * B_{kj}$ 
    end do
  end do
end do

```

where  $n_1 = k_1 m_1$ ,  $n_2 = k_2 m_2$  and  $n_3 = k_3 m_3$  with  $k_1$ ,  $k_2$  and  $k_3$  are integers greater than 1. The block operations  $C_{i,j} = C_{i,j} + A_{i,k} * B_{k,j}$  contain a reasonable amount of potential parallelism, so the algorithm proceeds by first partitioning the matrices and then dedicating the full resources of the machine to each of the block operations in turn.

The kernel algorithm for the block operation uses  $m_2$ -adic operations on each processor to accumulate  $p$  columns of  $C_{ij}$  at a time. (This should be a fairly portable approach since the multiply-add vector instruction tends to be the highest performance instruction on register-to-register vector processors.) Hence, the minimization of the  $T_a$  component of the algorithm time requires that:  $m_1$  is a multiple of the length of the vector registers (32 for a CEDAR cluster);  $m_2$  is large enough so that a significant fraction of the peak performance of the  $m_2$ -adic instruction is achieved (greater than 24 or so on a CEDAR cluster); and  $m_3$  is either large or a multiple of  $p$  ( $p = 8$  on a CEDAR cluster).

The minimization of the component  $T_l$  requires more effort. If it is assumed that  $A_{ik}$  is fetched from memory once and  $B_{kj}$  and  $C_{ij}$  are fetched as needed then the minimization of the number of loads is equivalent to minimizing  $\rho = m_1^{-1} + m_2^{-1}$  subject to the constraints  $m_2(p + m_1) \leq CS$ ,  $1 \leq m_1 \leq n_1$  and  $1 \leq m_2 \leq n_2$  where  $CS$  denotes the cache size. These constraints trace a rectangle and a hyperbola in the parameter plane. In [5] it is shown that the solution to the minimization problem, under reasonable assumptions including  $n_1 > \sqrt{CS}$ , is divided into two cases:

(1) If  $n_2(p + n_1) \leq CS$  then  $m_1 = n_1$  and  $m_2 = n_2$ . This yields the theoretical minimum for the number of loads for this primitive.

(2) If  $n_2(p + n_1) \geq CS$  then  $m_2 = \min(n_2, CS[\sqrt{CS} + p]^{-1})$  and  $m_1 = (CS/m_2) - p$ .

When this primitive is used as a rank- $n_2$  update it is reasonable to assume that  $n_2 \leq \sqrt{CS}$  and that  $n_1$  and  $n_3$  are large enough for the second case above to be applicable. The cache-miss ratio for this primitive is  $\mu \approx 1/2n_2$ . This implies that the relative cost of data loading is  $\lambda/(2n_2 + \lambda)$ . Clearly for BLAS2 where  $n_2 = 1$  the cost is too high. For a single CEDAR cluster it is reasonable to assume that  $\lambda \approx 3$  implying that 60 percent of the total time of the algorithm is spent loading data for the BLAS2 primitive while if, say, rank-32 updates are used this falls to an acceptable 4.5 percent. The difference in the performance of the basic primitives in the three levels of the BLAS on a single CEDAR cluster is dramatic. For example, experiments run at CSRD show that DAXPY for vectors of length  $n$  and a rank-1 update to an  $n$  by  $n$  matrix run at about 5.5 and 6.5 Megaflops respectively for  $n$  ranging from 256 to 1024. A rank-32

update on an  $n$  by  $n$  matrix runs at approximately 42 Megaflops for the same range of  $n$ .

**3. Matrix factorizations.** Many classical linear algebra algorithms can be expressed in terms of BLAS2 and BLAS3 primitives [1], [5]. In this section the LU decomposition and the modified Gram-Schmidt algorithm are considered as examples of the superiority of BLAS3 over BLAS2 on multivector processors with a hierarchical memory. In the following subsections the block algorithms are described and experimental results obtained on a single CEDAR cluster are presented.

**3.1. The LU decomposition.** The goal of the LU decomposition is to factor an  $n \times n$ -matrix  $A$  into the product of a lower triangular matrix  $L$  and an upper triangular matrix  $U$ . The classical LU factorization consists essentially of dotproducts, i.e., the classical BLAS or BLAS1, it can however also be expressed in terms of rank-1 updates, i.e., BLAS2 (see [3]).

In order to use the efficient BLAS3, a block LU factorization must be performed. This algorithm decomposes  $A$  into the products of block lower triangular matrix  $L_k$  and an block upper triangular matrix  $U_k$  with blocks of the size  $k \times k$  (it is assumed for simplicity that  $n$  is divisible by  $k$ ). Let  $A$  be a diagonally dominant matrix partitioned in the following way:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} I & 0 \\ L_{21} & I \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} \\ 0 & B \end{pmatrix}$$

where  $A_{11}$  is of order  $k$ .

The first step of the algorithm is

$$A_{11} \leftarrow A_{11}^{-1}, \quad A_{21} \leftarrow L_{21} = A_{21}A_{11}, \quad A_{22} \leftarrow B = A_{22} - L_{21}A_{12}.$$

The above computations are then performed recursively on the smaller matrix  $B$ . Note that the BLAS2 version of the classical LU decomposition is a special case of this algorithm for  $k = 1$ .

This block LU algorithm consists mainly of matrix-matrix operations. The only difficulty lies in inverting the  $k \times k$  blocks on the diagonal. This can be done by using the Gauss-Jordan algorithm which can be implemented efficiently on a CEDAR cluster. Such an algorithm is numerically stable for the diagonally dominant case [7]. Pivoting can be added to the block algorithm in a straightforward manner while maintaining performance superior to that of a BLAS2 implementation with pivoting [2]. This is due to the fact that the number of data loads required is not significantly affected by the addition of pivoting [5].

The block algorithm increases the number of operations by a factor of approximately  $(1 + 2k^2/n^2)$  over the classical LU factorization which requires about  $2n^3/3$  operations.

**3.2. A block Gram-Schmidt algorithm.** The goal of this algorithm is to factor an  $m \times n$ -matrix  $A$  into the product of an orthonormal  $m \times n$ -matrix  $Q$  and an upper rectangular  $n \times n$ -matrix  $R$  where  $m > n$  and  $A$  is of maximal rank.  $A$  is partitioned into two blocks  $A_1$  and  $B$  where  $A_1$  consists of  $s$  columns of order  $m$ , with  $Q$  and  $R$  partitioned accordingly.

$$(A_1, B) = (Q_1, P) \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix}.$$

The first step of the algorithm comprises the computations

$$A_1 = Q_1 R_{11}, \quad R_{12} = Q_1^T B, \quad B \leftarrow B_1 = B - Q_1 R_{12}.$$

The above computations are then performed recursively on the smaller problem  $B_1 = PR_{22}$ , etc. It is easily seen that this algorithm consists mainly of matrix operations. The second and third statements require approximately  $4ms(n-s)$  floating point operations while the first, the decomposition of  $A_1$  into the orthonormal matrix  $Q_1$  and the upper triangular matrix  $R_{11}$ , for which the modified Gram-Schmidt algorithm is used, requires only  $2ms^2$ . The BLAS2 version of the modified Gram-Schmidt algorithm is obtained by setting  $s$  to 1.

**3.3. Experimental results.** The classical method and the block variant of the LU factorization were implemented on a single CEDAR cluster using BLAS2 and BLAS3 primitives, respectively. Figure 1 illustrates the performance of both methods for systems of varying sizes. The block algorithm is about 3.5 times as fast as the classical LU factorization if  $n > 500$ .

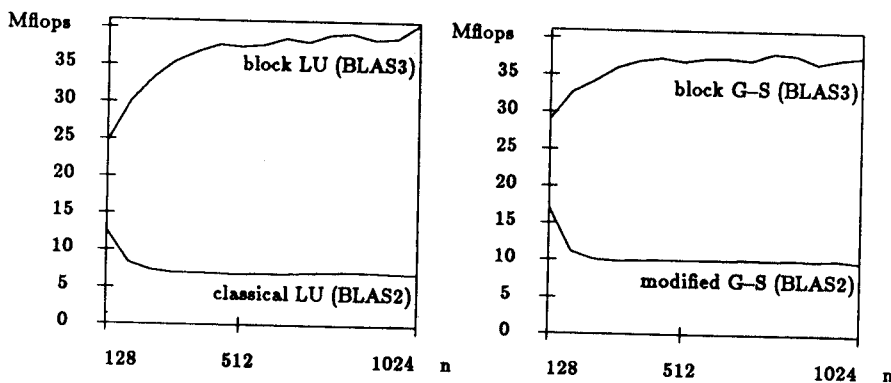


FIG. 1. Performance results.

Similar experiments were performed for the Gram-Schmidt orthogonal factorization algorithm. A BLAS2 version as well as a BLAS3 version of the algorithm with  $s = 32$  have been implemented on a single CEDAR cluster. The resulting performance is shown in Fig. 1. Increasing the order of  $A$  obviously improves the performance for the BLAS3 version as the matrix-matrix operations dominate the computation.

**4. Conclusion.** This correspondence has surveyed recent work at CSRD which has shown that a third level of the BLAS (BLAS3) based on matrix-matrix primitives is required in order to achieve high performance on multivector processors with a shared memory hierarchy. In this work the relative cost of data loading from memory to cache was expressed in terms of two generic systems parameters, the cache-miss and cost ratios. This model provides an intuitive and theoretically sound basis for analyzing the influence of the memory system on the performance of algorithms implemented on such architectures.

A numerical library based on the BLAS3 is currently under development at CSRD. Numerical experiments based on these algorithms have clearly shown the superiority of designing with BLAS3 primitives on systems such as CEDAR. The definition of BLAS3 is evolving as this work progresses, but at present some of the primitives are:  $C \leftarrow C \pm AB$ ;  $C \leftarrow C \pm A^T B$ ; triangular solvers for matrix equations  $LX = B$  and  $UX = B$ ; an inversion primitive using Gauss-Jordan reduction; and, of course, matrix decomposition primitives such as  $A = LU$ .

## REFERENCES

- [1] M. BERRY, K. GALLIVAN, W. HARROD, W. JALBY, S. LO, U. MEIER, B. PHILLIPS AND A. SAMEH, *Parallel numerical algorithms on the CEDAR system*, CSRD Report, CSRD University of Illinois at Urbana-Champaign, Urbana, IL, 1986.
- [2] D. CALAHAN, *Block-oriented, local-memory-based linear equation solution on the CRAY-2: uniprocessor algorithms*, Proc. ICPP 1986, IEEE Computer Society Press, Washington D.C., August 1986.
- [3] J. DONGARRA, J. BUNCH, C. MOLER AND G. W. STEWART, *LINPACK User's Guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1979.
- [4] J. DONGARRA, J. DUCROZ, S. HAMMARLING AND R. HANSON, *A proposal for an extended set of Fortran basic linear algebra subprograms*, Technical Memo #41, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, December 1984.
- [5] K. GALLIVAN, W. JALBY, U. MEIER AND A. SAMEH, *The impact of hierarchical memory systems on linear algebra algorithm design*, CSRD Report, CSRD University of Illinois at Urbana-Champaign, Urbana, IL, 1986.
- [6] W. JALBY AND U. MEIER, *Optimizing matrix operations on a parallel multiprocessor with a two-level memory hierarchy*, CSRD Report, CSRD University of Illinois at Urbana-Champaign, Urbana, IL, 1986.
- [7] G. PETERS AND J. WILKINSON, *On the stability of Gauss-Jordan elimination with pivoting*, Comm. ACM, 18 (1975), pp. 20-24.