CERN - Data Handling Division
DD/85/6
June 1985

Programming Tools for Linear Algebra

G.A. Erskine and H. Lipps

(Submitted to SIGNUM Newsletter)

PROGRAMMING TOOLS FOR LINEAR ALGEBRA

H. Lipps
G.A. Erskine

European Organisation for Particle Physics (CERN)
Geneva 23, Switzerland

Abstract

Dongarra et al have recently proposed extensions to the subprogram package BLAS to improve the performance of more elaborate linear algebra codes such as LINPACK. (See [2] - [4]) We summarize here our experience with a similar project.

1. Introduction

It has long been known that the use of Fortran-callable vector subroutines, e.g. to calculate a scalar product, will often not be efficient even if the vector subroutines are optimally coded in assembly language. We had to face this problem at CERN in 1975, when it was decided to supply our CDC and IBM computers with a compatible and efficient set of equation solvers and matrix inversion routines. For example, it was found that inversion of a real matrix of order 100 by a one-level Fortran program could be speeded up by a factor two on the CDC 7600 by calling an optimized scalar product routine, but could be speeded up by a factor of four if a one-level assembly language (COMPASS) routine was used for the job. Of course the timings depend heavily on the vector length of the problem and on the computer being used.

The algorithms for triangular factorization, elimination, and matrix inversion would normally be expressed in terms of scalar products, not only because this operation is faster than the equivalent step of Gaussian elimination on most computers, but also because the scalar products may be accumulated in double precision where machine architecture makes this easy - e.g. on IBM machines for data of types REAL and COMPLEX. However, once the algorithms have been expressed in this way, the necessary scalar products can all be grouped into products of a rectangular or triangular matrix by a vector. This formulation has the additional advantage that such products can be evaluated efficiently on vector computers that cannot accumulate individual scalar products efficiently.

A range of routines was therefore provided as part of the CERN program library to perform the necessary elementary operations. Based on these "Category 1" and "Category 2" routines, a third category of subroutines was implemented to provide matrix multiplication, triangular factorization, etc. The routines, 133 in all, have been available since 1979, not only for CDC and IBM machines, but also for most of the other computers used in Europe for particle physics.

Much preliminary work is necessarily involved in deciding upon the permissible kinds of data, the operations which shall be supported, the conventions to be followed by the calling sequences, etc. We shall give some examples in section 2 that will indicate the course which we have adopted, but will not address this subject in detail. However, the real difficulties begin when one tries to cope with the fact that portability and efficiency remain in conflict even when routines more powerful than those of the BLAS variety are called upon. We shall discuss some related implementation problems in sections 3 and 4.

2. What is provided

The CERN program library includes subprograms for linear algebra for data of three types. The first letter, *t* say, of each subprogram name is always one of the following, according to the type of data being processed:

<i>t</i> = R:	REAL
<i>t</i> = D:	DOUBLE PRECISION
<i>t</i> = C:	COMPLEX

Initially, we also provided for data types REAL*16 (t=Q) and COMPLEX*16 (t=W) for IBM machines, but we later abandoned these because they do not comply with the Fortran standard.

The package contains three categories of subprograms.

Category 1 subprograms provide vector operations like BLAS. The second letter of their names is always V. The remaining three or four letters define the operation; e.g. the function subprogram

tVMPY(N,X1,X2,Y1,Y2)

computes the inner product of two N-vectors x and y, whose first two elements are specified as actual arguments in the call.

Category 2 subprograms provide operations similar to those now proposed in [2]. The second letter of their names is M if they operate on rectangular (M,N) matrices, and U if they operate on upper triangular matrices. For instance, ten subroutines per data type exist to compute the following:

tMMPY: $z = Xy$	tUMPY: $z = Uy$
tMMPA: $z = Xy + z$	tUMPA: $z = Uy + z$
tMMPS: $z = Xy - z$	tUMPS: $z = Uy - z$
tMMNA: $z = -Xy + z$	tUMNA: $z = -Uy + z$
tMMNS: $z = -Xy - z$	tUMNS: $z = -Uy - z$

Sample calling sequences are

tMMPY(M,N,X11,X12,X21,Y1,Y2,Z1,Z2)

tUMPY(N,U11,U12,U22,Y1,Y2,Z1,Z2)

Here the rectangular matrix X and the upper triangular matrix U are each determined from three of their elements.

All vectors (other than working vectors for internal use) may have their elements spaced at any fixed distance; thus Y2 might coincide with Y1, or precede or follow it in storage. Similarly, the matrices need not be stored according to the Fortran convention; any equidistant spacing of their rows and columns is permitted. In particular, the same calling sequence will deal with transposed matrices. Similarly, lower triangular matrices may be substituted for upper triangular matrices.

Fortran implementation of this construct requires the use of a (non-standard) LOCF function, which the CERN library anyway supplies for those of our target machines that offer no equivalent. We think that this convention is so much more convenient to the user than one that relies on the specification of increments that the extra effort needed to cope with portability is justified.

Category 3 subprograms provide matrix multiplication, triangular factorization, inversion of full general or full symmetric matrices, linear equation solvers, and determinants. No sparse matrices of any kind are supported.

With the exception of the matrix multiplication routines, which again determine operand and result matrices from three of their elements to allow for transposition, the subroutines in this category expect matrices to be stored in accordance with the Fortran convention; e.g. subroutine

tEQN(N,A,IDIM,R,IFAIL,K,B)

replaces B by the solution X of the system of equations $AX = B$ of order N with K right-hand sides. Here IDIM is the leading Fortran dimension of A, R is a working vector, and IFAIL is set to zero if A is non-singular, otherwise to -1.

The routines for inversion and linear equations use triangular factorization with row interchanges. In the earlier versions the matrix was (implicitly) scaled to have rows of unit length, and a crude test for "near-singularity" (setting IFAIL = -2) was provided, based on the magnitude of the smallest pivot arising from this scaled matrix. We later removed this test because of the difficulty of providing a ma-

chine-independent check for it. We now feel that we made a mistake in doing so: users sometimes unknowingly generate matrices which are mathematically — but not numerically — singular, and want to be informed when this happens.

3. Testing

An elaborate test program has been developed to verify the package. Some of its tools have themselves become part of the library: copying or scaling of vectors and matrices, setting them to constant or random values, etc. For example, subroutine

tMRAN(M,N,A,B,Z11,Z12,Z21)

will fill the (M,N) matrix Z with random numbers from a uniform distribution with the limits A and B.

The arithmetic acceptance checks allow for variations due to round-off. This requirement is imposed by differences between equivalent algorithms. For example, to evaluate a real scalar product efficiently on CDC computers, one must accumulate the odd and even terms in different registers.

The logical checks go to some length to detect errors that arise mostly in assembly language programs: erroneous loop counts or addresses for example. An early version of the test program did not detect that the complex scalar product routine for CDC failed to set a B-register to 1. This was because the test program happened to leave the value 1 in this B-register whenever it called the scalar product routine. Since then we have made special provision for those systems, like CDC, which require the calling routine to save and restore its register contents whenever a subroutine is called. In these cases the test program places random values into all registers before calling the object of the test.

4. Implementation

European particle physics uses mostly scalar machines, especially IBM 370's, the CDC 7600 and its successors, and VAX computers. Vector machines have so far been used to only a limited extent. It is therefore essential that programs run efficiently on a wide range of scalar machines.

In particular, we must take into account those computers for which assembly language versions of our routines do not at present exist, and may perhaps never exist. For these machines, one-level Fortran routines are clearly preferable to two-level routines, even if Category 2 subroutines are being called. We must also take into account that many of the calculations involve fairly small linear systems, for which the administrative overhead of subroutine calls becomes significant. For this part of the work-spectrum two-level programs are inefficient even if assembly language routines are called. Further, our experience with various computers of IBM 370 architecture suggests that performance improvements resulting from the use of assembly language routines (of any complexity) range from zero to at best 25 percent. Assembly code that is optimal for one model of IBM 370 architecture will not be optimum for most others, and changes in the microcode can affect program performance at any time. Finally, even if optimum coding were possible in principle, it would require precise timing information, which the manufacturers do not supply. We have in fact provided IBM Assembler versions of our programs, but found that this involved a lot of work for little return.

For these reasons, the implementation of Category 3 programs via calls to subroutines of Categories 1 and 2 is often counter-productive. On the other hand, this procedure yields large gains on computers of the CDC 7600 variety, and of course on vector computers. Should the Fortran source code of Category 3 routines therefore exist in two versions in order to resolve the conflict? This would of course raise a number of administrative problems.

The implementation of Category 2 programs in assembly language presents many design problems that have no counterpart for Category 1. How is the product Xy to be computed for $(M,N)=(1,100)$ or $(2,100)$, or $(2,2)$, or $(100,1)$? How many special cases should be considered? What distinction, if any, should be made between columns of X stored consecutively and columns whose access would provoke memory bank conflicts? Optimized assembly code for a matrix-times-vector product is quite delicate to design, and is lengthy. Since one must produce several very similar algorithms, one obviously starts by developing a library of assembler macros from which various Category 1 and Category 2 routines may then be pieced together. Once the macro library is ready, it is about as easy to produce a one-level

equation solver from the macro library as it is to produce a Category 2 routine. Thus the same tools that provide optimized assembly language routines of Categories 1 and 2 can be used to provide efficient Category 3 routines at little extra cost. We thus conclude that for the standard algorithms of linear algebra, for which efficiency is important, two-level Fortran constructs are not attractive; and we therefore provide only one-level versions — in Fortran for all machines, in assembly language for some.

What then is the use of Category 1 and 2 routines? Hindsight would suggest that these routines can be useful in three ways. Firstly, as building blocks for miscellaneous algorithms that arise in application programs: reasonably efficient code may be developed at short notice. Secondly, for transient use on a new computer architecture while the underlying macro library is unavailable or incomplete. Thirdly, for use in checking out the macro library.

5. Conclusion

During the late seventies, a package of subprograms was developed at CERN to provide efficient linear algebra algorithms of varying complexity for a range of computing systems. Each routine is self-contained to avoid the overhead of subroutine calls. All exist in Fortran versions, and most in several assembly language versions.

6. References

- [1] Program Library Manual, CERN Computer Centre.
- [2] J.J. Dongarra, J. Du Croz, S. Hammarling, R.J. Hanson, A Proposal for an Extended Set of Fortran Basic Linear Algebra Subprograms, ACM Signum Newsletter, Vol. 20, No. 1, (January 1985), 2–18.
- [3] D.S. Dodson, J.G. Lewis, Issues Relating to Extension of the Basic Linear Algebra Subprograms, ACM Signum Newsletter, Vol. 20, No. 1, (January 1985), 19–22.
- [4] C.L. Lawson, R.J. Hanson, D.R. Kincaid, F.T. Krogh, Basic Linear Algebra Subprograms for Fortran Usage, ACM Transactions on Mathematical Software Vol. 5 (1979), 308–323.