A DESCRIPTION AND COMPARISON OF SUBROUTINES FOR COMPUTING EUCLIDEAN INNER PRODUCTS ON THE IBM 360

(inner ...)

BY

MICHAELA. MALCOLM

STAN-CS-70-175

OCTOBER 1970

COMPUTER SCIENCE DEPARTMENT

School of Humanities and Sciences

STANFORD UNIVERSITY

A DESCRIPTION AND COMPARISON OF SUBROUTINES FOR COMPUTING EUCLIDEAN INNER PRODUCTS ON THE IBM 360

by

Michael A. Malcolm

I. Introduction

In many algorithms, a Euclidean inner product of two vectors must be computed with greater precision than the rest of the calculations. An example is the calculation of the residual vector

$$r = b - A\hat{x} \tag{1}$$

used in an algorithm for improving an approximate solution $\hat{x}$ of the linear system

$$Ax = b \quad .$$

When the inner product occurs in an algorithm coded in short precision, it is usually sufficient to accumulate it in long precision (double precision). Long-precision arithmetic is a hardware feature of many machines; if so, the inner product is easily coded and quickly executed. However, when long-precision arithmetic is not available, or when the entire algorithm is coded in long precision, the inner product routine becomes more difficult to code and execution time may become excessive.

This report is primarily concerned with existing routines for evaluating inner products using more precision than long, for use within long-precision programs for the IBM System/360. Several such subroutines can be called from Fortran H programs; one is available for Watfor (or Watfiv) For-bran programs and one for Algol W.

II. Algol w

The double precision inner product routine available for Algol W programs is the

```
long real procedure ip2 (integer i; integer value ℓ, s, u;
     long real a, b; long real value c);
comment This procedure computes the sum of products a×b and
     adds it to the extra term c.  The bound variable i is used
     to indicate the subscript in the components of the vectors
     a and b over which the scalarproduct is formed.  Although the
     procedure body is more complicated, it can be illustrated as
     follows:
     begin long real sum, sum := 0.0L,
          for i := ℓ step s until u do sum := sum + a*b,
          sum + c
     end;
```

Jensen's device is used through the bound variable i . For example, ip2 could be used to compute the vector $\underset{\sim}{r}$ in Equation (1) as follows:

```
for i := 1 step 1 until n do
r(i) := -ip2(k,1,1,n,A(i,k),x(k),-b(i))
```

Since each product has 28 significant hex digits and a double word has only 14 digits, a technique related to that suggested by Møller [1965] is used to retain full significance. For illustrative purposes, consider the following segment of an Algol W program:

```
real t; long real a, a1, a2, b, b1, b2, b3;
.
comment a and b have been assigned double precision values;
t := a; a1 := t; a2 := a-a1;
t := b; b1 := t; b3 := b - b1;
t := b3; b2 := t; b3 := b3 - b2;
.
.
```

The above program segment splits the numbers a and b so that

$$a = a1 + a2$$

$$b = b1 + b2 + b3 \ .$$

Thus

$$a \times b = (a1 + a2) \times (b1 + b2 + b3)$$

$$= a1*b1 + a1*(b2 + b3) + a2*b1 + a2*b2 + a2*b3 \qquad (2)$$

where * indicates double-precision floating-point multiplication and the symbols x , + and = have the usual mathematical interpretation.

The terms of Equation (2) are accumulated using a technique suggested by Malcolm [1970]. It follows directly from Theorem 2 in Malcolm [1970] that provided n < 13107 , the result $(\hat{\xi})$ calculated by ip2 satisfies

$$\xi = \hat{\xi}(1 + \varepsilon) \qquad (3)$$

where

$$|\varepsilon| \leq 4 \cdot 16^{-12}$$

and ξ is the exact result. The procedure can be easily modified to accommodate n $\geq$ 13107 and still satisfy Equation (3).

The parameters i , a and b are passed by name to give maximum generality. One may wish to modify this to economize on execution time.

III. Watfor (or Watfiv) Fortran

The same techniques used in ip2 are implemented in two Fortran subroutines: DPPUT(A,B) and IPTOTL(S) . The call:

```
CALL DPPUT(A,B)
```

adds the product A x B (A and B are double precision) to the accumulators. The call:

```
CALL IPTOTL(S)
```

sums the accumulators and assigns the long precision result to S . The subroutine IPTOTL leaves the accumulators in their initial state (all zero).

The result S (= $\hat{\xi}$) satisfies (3) provided DPPUT has not been called more than 13,107 times since the accumulators were last initialized.

DPPUT and IPTOTL use a named common area called DPACCC for storing the accumulators. A BLOCK DATA subprogram is used for initializing the named common data area.

Following is an example using DPPUT and IPTOTL to calculate the r vector in Equation (1).

```
      DØ 10 I = 1,N
      DØ  5 J = 1,N
    5 CALL DPPUT(-A(I,J),X(J))
      CALL DPPUT(B(I),1.0D0)
   10 CALL IPTOTL(R(I))
```

IV. Fortran H

Several efficient subroutines can be called by a Fortran H program for computing double-plus inner products.

A. VPR2

VPR2 is a subroutine written by Ehrman [1967] that forms the double-long product of two double precision arguments and adds it to a double-long sum. For example, VPR2 could be used for computing the r vector of Equation (1) as follows:

```
      REAL*8 R1(2),A(N),B(N),X(N),R(N)
      INTEGER IEXP
      .
      .
      DØ 10 I = 1,N
      IEXP = 0
      R1(1) = 0.0D0
      R1(2) = 0.0D0
      DØ 5 J = 1,N
    5 CALL VPR2(-A(I,J),X(J),R1(1),IEXP)
      CALL VPR2(1.0D0,B(I),R1(1),IEXP)
      IF (IEXP.NE.0) GØ TØ 100
   10 R(1) = R1(1)
      .
      .
      .
  100 (write error message and/or terminate)
```

In the above example, R1 is an accumulator with 30 hex digits (two double words with the exponent) and IEXP is used as an indication of underflow or overflow.

Although VPR2 uses a 30 hex digit accumulator, it can still result in a large relative error. Examples can be constructed that result in no significant digits. However,-practical algorithms in which this phenomenon causes an unacceptable loss of precision are probably rare.

All calculations in VPR2 are performed in the "general registers". Although VPR2 requires a subroutine linkage for each term of the inner product, execution times compare favorably with the fastest routines.

B. DPPUT and IPTOTL

The routines described in Part III for use in Watfor are available in more efficient versions coded in PL360 for use with Fortran H. The PL360 versions of DPPUT and IPTOTL differ from the Fortran versions in that full precision accuracy is obtained and the result is correctly rounded. This is achieved by a technique described in Section V of Malcolm [1970]. Also, the result has full precision accuracy and is correctly rounded.

C. DPDOTP

DPDOTP is a PL360 function subroutine which uses the same techniques as DPPUT and IPTØTL described above. The function call for DPDOTP has a variable length parameter list. The full formal parameter list is:

DPDOTP(A,B,N,XTERM,INCA,INCB,PVA,PVB)

where

A,B -- The locations of the first components of the long-precision vectors to be multiplied

N -- The number of terms entering the inner product

XTERM -- An extra double precision term to be added to the inner product (optional)

INCA -- Number of (double) words separating successive elements of the vector' A (optional)

INCB -- Number of (double) words separating successive elements of the vector B (optional)

PVA -- Integer vector specifying a permutation of the elements of the vector A (optional)

PVB -- Integer vector specifying a permutation of the elements of the vector B (optional)

In the actual parameter list, only the first three parameters (A , B and N) are required. Default values of the remaining parameters are:

```
XTERM  = 0.0D0
INCA   = 1
INCB   = 1
PVA(I) = I      (I = 1,2,...)
PVB(I) = I      (I = 1,2,...)
```

For illustrative purposes assume the following declarations

```
REAL*8 DPDOTP,A(N,N),B(N),C(N),SUM,R(N),X(N)
INTEGER*4 PA(N)
```

Note that DPDOTP must be declared as a long-precision floating-point variable. A statement which sets SUM to the inner product of the vectors B and C is

```
SUM= DPDOTP(B,C,N)
```

Another example is the calculation of the residual vector in Equation (1):

```
      DØ 10 I = 1,N
10 R(1) = -DPDØTP(A,X,N,-B(I),N)
```

In this example, INCA must be N because Fortran stores the array A in column order (see the Fortran IV(H) Programmer's Guide) which means neighboring elements in a given row of A are separated by N double words. If the **columns** of A , in the above example, were permuted as

specified by the integer vector PA , the calculation of the residual vector would then be as follows:

```
      DØ 10 I = 1,N
   10 R(1) = -DPDØTP(A,X,N,-B(I),N,1,PA)
```

A PL360 single precision function subroutine for calculating the exact rounded inner product of single precision vectors is also available. This routine, called SPDOTP, has the same calling sequence as DPDOTP.

D. DOTP

DOTP is an Assembler Language function subroutine written at Argonne National Laboratories (see Jordan [1967]). The formal parameter list is

```
DOTP(A,B,N)
```

where

A, B -- The locations of the first components of the vectors to be multiplied

N -- The number of terms entering the inner product

For example, the residual vector in Equation (1) could be calculated as follows:

```
      REAL*8 DØTP,A(N,N),X(N),B(N),R(N),TEMP(N)
      .
      .
      .
      DØ 10 I = 1,N .
      DØ  5 J = 1,N
    5 TEMP(J) = A(I,J)
   10 R(1)  = B(1) - DØTP(TEMP,X,N)
```

Note that DOTP must be declared as a long-precision variable.

DOTP uses the same techniques as DPDOTP (i.e., splitting the operands and 32 accumulators); however, DOTP does a number of internal subroutine linkages (proportionalto N) to code that is in line in DPDOTP.

V. Comparison of Execution Times

Each of the routines described above has undergone extensive tests to insure accuracy. In addition to these tests, each routine was timed on the 360/67 with the following two calculations:

Test No. 1: $\sum_{k=1}^{N} a_{ik} \times b_k$

Test No. 2: $\sum_{k=1}^{N} a_k \times b_k$

Each factor a_{ik} , a_k , b_k entering the inner product for these tests was equal to 3.1415926535897932 .

The experimental results are tabulated in Table I in terms of values of K for determining execution time according to

execution time = K x N

in milliseconds.

The people who programmed the various routines are acknowledged in Table I.

TABLE I

Values of K for

execution time = K x no. of terms in inner product (ms) */

Calling Language	Inner Product Routine	Inner Product Compiler	Programmer	K for $\sum_k a_{ik} \times b_k$	K for $\sum_k a_k \times b_k$
Algol W	'ip2	Algol W (w/o $NOCHECK)	Michael Saunders	0.710	0.703
Algol w	ip2	Algol W (with $NOCHECK)	Michael Saunders	0.544	0.526
Fortran	DPPUT IPTOTL	Watfiv (w/o NOCHECK)	Gordon Gullahorn	2.12	2.03
Fortran	DPPUT IPTOTL	Watfiv (with NOCHECK)	Gordon Gullahorn	2.11	2.06
Fortran	DPPUT IPTOTL	Fortran H opt = 0	Gordon Gullahorn	0.424	0.421
Fortran	DPPUT IPTOTL	Fortran H opt = 2	Gordon Gullahorn	0.332	0.332
Fortran	DPPUT IPTOTL	PL360	Michael Malcolm	0.212	0.210
Fortran	DPDOTP	PL360	Michael Malcolm	0.184	0.184
Fortran	VPR2	OS/Assembler	John Ehrman	0.196	0.196
Fortran	DOTP	OS/Assembler	D. Jordan	0.242	0.218

*/ All tests were performed on an IBM 360/67.

VI. Conclusions

Many long-precision routines requiring accurate inner products can be coded in either Fortran or Algol W. For Fortran, DPPUT and IPTOTL are probably the most useful for three reasons: (1) they are easy to use and fast; (2) accuracy of the result is guaranteed; and (3) programs using them can be debugged and run with the Watfor (or Watfiv) compiler. For programs which are to be debugged and run with the Fortran H compiler, DPDOTP is probably the best because it is easy to use, execution time is minimal and the result is guaranteed.

Bibliography

Ehrman, John [1967]. "Double-Double Accumulation of Inner Products." Stanford University Computation Center, Extrinsic Program Library No. C 003.

Fortran IV(H) Programmer's Guide. IBM System/360 Operating System. File No. S360-25, Form C28-6602-3.

Jordan, D. F. [1967]. ANL-F1545-DOTP, "Extra-Precision Accumulating Inner Product." Argonne National Laboratory, Applied Mathematics Division. System/360 Library Subroutine. Argonne, Illinois. November.

Malcolm, Michael [1970]. "An Algorithm for Floating-Point Accumulation of Sums with Small Relative Error." Technical Report No. STAN-CS-70-163. Computer Science Department, Stanford University. June.

Møller, Ole [1965]. "Quasi Double-Precision in Floating-Point Addition." BIT 5. 37-50.