Beyond Fortran77 : Pointers, Dynamic Memory Allocation Efficient Programming with BLAS

Jürg Hutter
MPI für Festkörperforschung
Stuttgart

April 11, 1995

Beyond Fortran77

Pointers

Pointers are an extension to the Fortran77 standard implemented in most FORTRAN compilers. The pointer type of AIX FORTRAN (and CRAY FORTRAN) is also called integer pointer to distinguish this type from the "true" pointers in Fortran 90. Details about integer pointers can be found in the AIX XL Fortran Compiler/6000 Language Reference manual.

(integer) Pointer statement
POINTER (int_pointer,pointee)

int_pointer integer variable, in CPMD all integers that are used as pointers have a name of type IP_{pointee}.

pointee variable name, array declaration

Example:

```
      subroutine sub1(a,n)
      dimension a(n),b(2,*)
      pointer (ip_b,b)
c..associate a with b
      ip_b = loc(a(1))
      ...
```

loc(a) is an intrinsic function that returns the address of variable a. This allows for a flexible use of arrays (much more than the clumsy EQUIVALENCE statement in standard Fortran 77.

Warning: It is easy to make a program unreadable with pointers.

Warning: CRAY compilers assume (for optimization reasons) that no two pointers are pointing to the same address.

Warning: Pointers slow down your Fortran program.

Dynamic Memory Allocation

In Fortran77 all array definitions have to be done with constants or parameters.
Example

```
parameter (maxat=100)
dimension coor(3,maxat)
```

The size of all arrays is fixed at compile time.
Advantage: allows for high optimization.
Disadvantage: the program is not flexible (waste of memory, multiple versions, etc.)

Memory Management System(MMS)

C language routines : malloc, free (some Fortran dialects provide routines with similar functionality, e.g. CRAY hpalloc, hpdealloc)

malloc allocates memory, if necessary gets new memory from the operating system.

free returns memory, (IBM MMS: gives memory back to MMS, never to the operating system, programs only grow, never shrink)

Calling malloc and free from Fortran programs

```
IP_A = MALLOC(%VAL(length_of_a))
```

IP_A integer pointer

length_of_a number of **bytes** to be allocated.

%VAL tells the compiler that malloc needs the value of length_of_a, not its address. (Fortran: call by reference; C: call by value)
This is IBM specific, may be different on other computers.

```
CALL FREE(%VAL(IP_A))
```

The MMS 'knows' (this means, it keeps a list) how much memory is associated with pointer IP_A, but only if the value of IP_A was calculated with malloc and subsequently not changed.

CPMD GOLDEN RULE 1 :

Never use a variable with name IP_{name} for arithmetics. The only context in which such a variable appears on the left side of an equal sign, is as result of the malloc and loc functions.

Example:
Subroutine with adjustable local scratch space.

```
      subroutine mult3(a,b,c,d,n)
c..calculates d=a*b*c, where a,b,c,d are n by n matrices
      implicit none
      real*8 a(n,*),b(n,*),c(n,*),d(n,*)
      integer n
c..local variables
      integer maxn,ip_scr,len
      real*8 scr(n,*)
      pointer (ip_scr,scr)
      data ip_scr,maxn /0,0/
      save ip_scr,maxn
c..memory management for local scratch
      if(n.gt.maxn) then
        if(ip_scr.ne.0) call free(%val(ip_scr))
        len=8*n*n
        ip_scr=malloc(%val(len))
        maxn=n
      endif
c..two successive matrix multiplies
      call dgemm('N','N',n,n,n,1.0d0,a,n,b,n,0.0d0,scr,n)
      call dgemm('N','N',n,n,n,1.0d0,scr,n,c,n,0.0d0,d,n)
      return
      end
```

Dynamic Memory allocation in CPMD

Integer function **MEMORY(NW)**

- allocates nw (IBM) double-words of memory

```
C     ==================================================================
      FUNCTION MEMORY(LENGTH)
      IMPLICIT REAL*8 (A-H,O-Z)
      INCLUDE 'irat.inc'
C.....................................................................
C     FILE irat.inc
C  64/(NUMBER OF BITS PER INTEGER)
C     PARAMETER(IRAT=2)
C.....................................................................
      POINTER (IP_XM,XM(*)),(IP_IXM,IXM(*))
C     ==--------------------------------------------------------------==
C     ==  ALLOCATION OF LENGTH * REAL*8 WORDS OF MEMORY               ==
C     ==--------------------------------------------------------------==
#ifdef YMP
      INTEGER ERRCODE
      CALL HPALLOC(IP_XM,LENGTH,ERRCODE,0)
      IF(ERRCODE.NE.0) THEN
        WRITE(6,*) ' ALLOCATION OF ',LENGTH,' WORDS OF MEMORY FAILED '
        CALL STOPGM('MEMORY',' ')
      ENDIF
#else
      LEN = 8 * LENGTH
      LEN=MAX(LEN,400)
      IP_XM=MALLOC(%VAL(LEN))
      IF(IP_XM.LE.0) THEN
        WRITE(6,*) ' ALLOCATION OF ',LENGTH,' WORDS OF MEMORY FAILED '
        CALL STOPGM('MEMORY',' ')
      ENDIF
      IP_IXM=LOC(XM)
      DO I=1,LENGTH*IRAT
        IXM(I)=0
      ENDDO
#endif
      MEMORY=IP_XM
C     ==--------------------------------------------------------------==
      RETURN
      END
C     ==================================================================
```

subroutine **FREEM(IP_name)**

- releases memory associated with integer pointer IP_name

```
C     ==================================================================
      SUBROUTINE FREEM(IPOINT)
C     ==--------------------------------------------------------------==
C     ==  FREE MEMORY ASSOCIATED WITH POINTER IPOINT                  ==
C     ==--------------------------------------------------------------==
#ifdef YMP
      CALL HPDEALLC(IPOINT,ERRCODE,0)
      IF(ERRCODE.NE.0) THEN
        WRITE(6,*) ' FREEM | Error in deallocation of memory '
        CALL STOPGM('FREEM',' ')
      ENDIF
#else
      CALL FREE(%VAL(IPOINT))
#endif
C     ==--------------------------------------------------------------==
      RETURN
      END
C     ==================================================================
```

Hard coded upper limits in CPMD

MAXSP Maximum number of atomic species (20).

NHX Maximum number of Gauss-Hermit points (sum over all angular momenta). (100)
Maximum number of projectors for separable pseudo-potentials.
But NHXS: actual maximum of the quantities mentioned above.

LX $2 * L_{max} - 1$, where L_{max} is the maximum angular momenta + 1. (5)

MMAXX : Maximum number of points for radial meshes in real space. (999)

Quantities calculated dynamically

NSX Number of atomic species.

NAX Maximum umber of atoms per species.

NAT Total number of atoms.

NGW Number of plane-waves for wavefunctions.

NHG Number of plane-waves for density.

NR1,NR2,NR3 Number of grid points in x, y, and z direction.

N Number of states, but in most routines used as dummy variable **NSTATE**.

Basic Linear Algebra Subprograms (BLAS)

BLAS are an industry-wide standard, providing a uniform functionality and call sequence interface, which makes an application using calls to BLAS highly portable across high-performance platforms from different vendors. There are $\approx$ 140 different subroutines in BLAS. BLAS are a subset of ESSL (Engineering and Scientific Subroutine Library) of IBM.

Warning : ESSL contains many BLAS-like subroutines that are not standard.

BLAS is divided in functions and subroutines for :

- vector-scalar operations (BLAS 1)
- matrix-vector operations (BLAS 2)
- matrix-matrix operations (BLAS 3)

Subroutines starting with

- 'S' are for single precision real arithmetic (32 bit on IBM, but 64 bit on CRAY computers)
- 'D' are for double precision real arithmetic
- 'C' are for single precision complex arithmetic
- 'Z' are for double precision complex arithmetic

The most important BLAS routines

- Level 1 BLAS

  DDOT inner product of two vectors
  $s \leftarrow \vec{x}^{\dagger}\vec{y}$

  DAXPY scale a vector by a scalar and add to another vector
  $\vec{y} \leftarrow \vec{y} + \alpha\vec{x}$

  DCOPY copy a vector onto another vector
  $\vec{y} \leftarrow \vec{x}$

  DSCAL scale a vector by a scalar
  $\vec{y} \leftarrow \alpha\vec{y}$

- Level 2 BLAS

  DGEMV scale a matrix-vector product by a scalar and add it to another scaled vector
  $\vec{y} \leftarrow \beta\vec{y} + \alpha\mathbf{A}\vec{x}$

  DGER rank 1 update of a matrix
  $\mathbf{A} \leftarrow \mathbf{A} + \alpha\vec{x}\vec{x}^{\dagger}$

- Level 3 BLAS

  DGEMM add a matrix product to another matrix
  $\mathbf{C} \leftarrow \beta\mathbf{C} + \alpha\mathbf{AB}$

CPMD GOLDEN RULE 2 :

Use BLAS whenever possible, you get performance and portability for no cost (this is almost free lunch).

NETLIB

Netlib is a collection of mathematical software, papers and databases.

How to access netlib

- World Wide Web
  http://www.netlib.org/idex.html/
  only restricted access for severs from outside the US
- anonymous ftp
  netlib2.cs.utk.edu (128.169.92.17)
- e-mail to netlib at ornl.gov or netlib at nac.no
  send a message with text
  send index

What you can find in netlib (minimal selection)

- Source code for all BLAS routines
- Benchmarks for almost all available computers (present and past)
- Fortran source codes for special functions (e.g. Bessel functions)
- f2c; a Fortran to C converter
- a package for doing spline interpolations (now also part of CPMD)
- LINPACK and EISPACK
- LAPACK (used in CPMD to replace several calls to ESSL routines for non-IBM computers)

Examples of usage of BLAS in CPMD

Function DOTP: calculate the overlap of two wavefunctions.

```
C     ==================================================================
      FUNCTION DOTP(N,A,B)
C     ==--------------------------------------------------------------==
      IMPLICIT REAL*8 (A-H,O-Z)
      LOGICAL GEQ0
      COMMON/GEQ0L/GEQ0
      DIMENSION A(*),B(*)
C     ==--------------------------------------------------------------==
      DOTP=2.0D0*DDOT(2*N-2,A(3),1,B(3),1)
      IF(GEQ0) THEN
        DOTP=DOTP+A(1)*B(1)
      ELSE
        DOTP=DOTP+2.0D0*(A(1)*B(1)+A(2)*B(2))
      ENDIF
C     ==--------------------------------------------------------------==
      RETURN
      END
C     ==================================================================
```

Subroutine OVLAP: Computes the overlap matrix between two sets of wavefunctions.

```
C     ==================================================================
      SUBROUTINE OVLAP(NSTATE,A,C1,C2)
C     ==--------------------------------------------------------------==
C     ==         COMPUTES THE OVERLAP MATRIX A = < C1 | C2 >          ==
C     ==--------------------------------------------------------------==
      IMPLICIT REAL*8 (A-H,O-Z)
      INCLUDE 'system.h'
      INCLUDE 'geq0.inc'
      INCLUDE 'spin.inc'
      COMPLEX*16 C1(NGW,*),C2(NGW,*)
      DIMENSION A(NSTATE,*)
C     ==--------------------------------------------------------------==
      CALL TISET('     OVLAP',ISUB)
      IF(TLSD) THEN
        CALL AZZERO(A,NSTATE*NSTATE)
C..Alpha spin
        CALL DGEMM('T','N',NSUP,NSUP,2*NGW,2.0D0,C1(1,1),2*NGW,
     *             C2(1,1),2*NGW,0.0D0,A(1,1),NSTATE)
        IF(GEQ0)
     *  CALL DGER(NSUP,NSUP,-1.0D0,C1(1,1),2*NGW,
     *             C2(1,1),2*NGW,A(1,1),NSTATE)
C..Beta spin
        CALL DGEMM('T','N',NSDOWN,NSDOWN,2*NGW,2.0D0,C1(1,NSUP+1),2*NGW,
     *             C2(1,NSUP+1),2*NGW,0.0D0,A(NSUP+1,NSUP+1),NSTATE)
        IF(GEQ0)
     *  CALL DGER(NSDOWN,NSDOWN,-1.0D0,C1(1,NSUP+1),2*NGW,
     *             C2(1,NSUP+1),2*NGW,A(NSUP+1,NSUP+1),NSTATE)
      ELSE
C..LDA
        CALL DGEMM('T','N',NSTATE,NSTATE,2*NGW,2.0D0,C1,2*NGW,C2,2*NGW,
     *             0.0D0,A,NSTATE)
        IF(GEQ0)
     *  CALL DGER(NSTATE,NSTATE,-1.0D0,C1,2*NGW,C2,2*NGW,A,NSTATE)
      ENDIF
      CALL TIHALT('     OVLAP',ISUB)
C     ==--------------------------------------------------------------==
      RETURN
      END
C     ==================================================================
```

Subroutine XGRAD: Calculates the derivative of the total energy with respect to orbital rotations. [1]

$$\begin{aligned}\frac{\partial E(\mathbf{C_o}(\mathbf{X}))}{\partial \mathbf{X}} &= \mathbf{R}^\dagger\Big(-\Big(\mathbf{R}(\mathbf{G}^{(1)^\dagger}+\mathbf{G}^{(1)})\mathbf{R}^\dagger\Big)\otimes\mathbf{\Gamma}^{(1)} \\ &\quad -\Big(\mathbf{R}(\mathbf{X}\mathbf{G}^{(2)}+\mathbf{G}^{(2)^\dagger}\mathbf{X}^\dagger)\mathbf{R}^\dagger\Big)\otimes\mathbf{\Gamma}^{(2)}\Big)\mathbf{R}\mathbf{X} \\ &\quad -\mathbf{R}^\dagger\mathbf{\Gamma}^{(3)}\mathbf{R}\mathbf{G}^{(2)^\dagger}\end{aligned}$$

```
C     ==================================================================
      SUBROUTINE XGRAD(X0,X2,C2,RMAT,REIG,LDX,NGW,NSTATE,SCR,LSCR)
C     ==--------------------------------------------------------------==
      IMPLICIT REAL*8 (A-H,O-Z)
      COMPLEX*16 C2(NGW,*)
      DIMENSION X2(*),RMAT(NSTATE,*),REIG(*),SCR(*),X0(*)
C     ==--------------------------------------------------------------==
C     ==  CALCULATE THE GRADIENT OF THE ENERGY WITH RESPECT TO        ==
C     ==  ORBITAL ROTATIONS                                           ==
C     ==                                                              ==
C     ==  INPUT  : C2     ; GRADIENT WRT COEFFICIENTS                 ==
C     ==         : RMAT*REIG*RMAT(T) = X*X(T)                         ==
C     ==  OUTPUT : X2     ; GRADIENT WRT ROTATIONS                    ==
C     ==--------------------------------------------------------------==
      CALL TISET('     XGRAD',ISUB)
      NLDX=2*NGW-1-NSTATE
      IC1    =  1
      IC2    =  IC1 + NSTATE*NSTATE
      IC3    =  IC2 + NSTATE*NLDX
      IC4    =  IC3 + NSTATE*NSTATE
      IC5    =  IC4 + NSTATE*NSTATE
      ICTOP  =  IC5 + NSTATE*NSTATE
      IF(ICTOP.GT.LSCR) THEN
        WRITE(6,*) ' XGRAD: SCRATCH SPACE TOO SMALL BY :',ICTOP-LSCR
        CALL STOPGM('XGRAD',' SCRATCH SPACE')
      ENDIF
C     ==--------------------------------------------------------------==
C     ==  DECOMPOSE THE GRADIENT                                      ==
C     ==--------------------------------------------------------------==
      SQ2=DSQRT(2.0D0)
      L=0
      K=0
```

[1] J. Hutter, M. Parrinello, and S. Vogel, J. Chem. Phys. **101**, 3862 (1994)

```
      DO J=1,NSTATE
        DO I=2,NSTATE+1
          SCR(IC1+K)=SQ2*DREAL(C2(I,J))
          K=K+1
        ENDDO
        DO I=NSTATE+2,NGW
          SCR(IC2+L)=SQ2*DREAL(C2(I,J))
          L=L+1
        ENDDO
        SCR(IC2+L)=-DREAL(C2(1,J))
        L=L+1
        DO I=2,NGW
          SCR(IC2+L)=SQ2*DIMAG(C2(I,J))
          L=L+1
        ENDDO
      ENDDO
C     ==--------------------------------------------------------------==
C     ==  F2=R*X*G2*R(T)                                              ==
C     ==--------------------------------------------------------------==
      CALL DGEMM('T','N',NSTATE,NSTATE,NLDX,1.0D0,X0,LDX,SCR(IC2),
     *           NLDX,0.0D0,SCR(IC3),NSTATE)
      CALL DGEMM('T','N',NSTATE,NSTATE,NSTATE,1.0D0,RMAT,NSTATE,
     *           SCR(IC3),NSTATE,0.0D0,SCR(IC4),NSTATE)
      CALL DGEMM('N','N',NSTATE,NSTATE,NSTATE,1.0D0,SCR(IC4),NSTATE,
     *           RMAT,NSTATE,0.0D0,SCR(IC3),NSTATE)
C     ==--------------------------------------------------------------==
C     ==  F1=R*G1*R(T)                                                ==
C     ==--------------------------------------------------------------==
      CALL DGEMM('T','N',NSTATE,NSTATE,NSTATE,1.0D0,RMAT,NSTATE,
     *           SCR(IC1),NSTATE,0.0D0,SCR(IC4),NSTATE)
      CALL DGEMM('N','N',NSTATE,NSTATE,NSTATE,1.0D0,SCR(IC4),NSTATE,
     *           RMAT,NSTATE,0.0D0,SCR(IC1),NSTATE)
C     ==--------------------------------------------------------------==
C     ==  F1+F1(T) ; F2+F2(T)                                         ==
C     ==--------------------------------------------------------------==
      DO I=1,NSTATE
        DO J=I,NSTATE
          IJ=(I-1)*NSTATE+J-1
          JI=(J-1)*NSTATE+I-1
          F1=SCR(IC1+IJ)+SCR(IC1+JI)
          F2=SCR(IC3+IJ)+SCR(IC3+JI)
          SCR(IC1+IJ)=F1
          SCR(IC1+JI)=F1
          SCR(IC3+IJ)=F2
```

```
          SCR(IC3+JI)=F2
        ENDDO
      ENDDO
C     ==--------------------------------------------------------------==
C     ==  C1; C2                                                      ==
C     ==--------------------------------------------------------------==
      K=0
      DO I=1,NSTATE
        DO J=1,NSTATE
          A=DSQRT(REIG(I))
          B=DSQRT(REIG(J))
          A1=0.5D0*(A+B)
          B1=0.5D0*(A-B)
          SCR(IC4+K)=0.5D0*SBES0(A1)*SBES0(B1)
          SCR(IC5+K)=FC4(A,B)
          K=K+1
        ENDDO
      ENDDO
C     ==--------------------------------------------------------------==
C     ==  FC = - F1 x C1 - (F2 x C2)(T) + F2 x C2                     ==
C     ==--------------------------------------------------------------==
      K=0
      DO I=1,NSTATE
        DO J=1,NSTATE
          SCR(IC1+K)=-SCR(IC1+K)*SCR(IC4+K)-SCR(IC3+K)*SCR(IC5+K)
          K=K+1
        ENDDO
      ENDDO
C     ==--------------------------------------------------------------==
C     ==  X2 = R(T) * FC * R * X - J0(P)*G2(T)                        ==
C     ==--------------------------------------------------------------==
      CALL DGEMM('N','N',NSTATE,NSTATE,NSTATE,1.0D0,RMAT,NSTATE,
     *           SCR(IC1),NSTATE,0.0D0,SCR(IC3),NSTATE)
      CALL DGEMM('N','T',NSTATE,NSTATE,NSTATE,1.0D0,SCR(IC3),NSTATE,
     *           RMAT,NSTATE,0.0D0,SCR(IC1),NSTATE)
      CALL DCOPY(NSTATE*NSTATE,RMAT,1,SCR(IC3),1)
      DO I=1,NSTATE
        DO J=1,NSTATE
          RMAT(I,J)=RMAT(I,J)*SBES0(DSQRT(REIG(J)))
        ENDDO
      ENDDO
      CALL DGEMM('N','T',NSTATE,NSTATE,NSTATE,1.0D0,RMAT,NSTATE,
     *           SCR(IC3),NSTATE,0.0D0,SCR(IC4),NSTATE)
      CALL AZZERO(X2,LDX*NSTATE)
```

```
      CALL DGEMM('N','T',NLDX,NSTATE,NSTATE,-1.0D0,SCR(IC2),NLDX,
     *           SCR(IC4),NSTATE,0.0D0,X2,LDX)
      CALL DGEMM('N','T',NLDX,NSTATE,NSTATE,1.0D0,X0,LDX,
     *           SCR(IC1),NSTATE,1.0D0,X2,LDX)
C     ==--------------------------------------------------------------==
C     ==  NORM AND MAX. ELEMENT OF GRADIENT                           ==
C     ==--------------------------------------------------------------==
      IMAXE=IDAMAX(NSTATE*NLDX,X2,1)
      GEMAX=ABS(X2(IMAXE))
      CNORM=DSQRT(DDOT(NSTATE*LDX,X2,1,X2,1)/(NSTATE*NLDX))
      CALL TIHALT('     XGRAD',ISUB)
C     ==--------------------------------------------------------------==
      RETURN
      END
C     ================================================================
```

Homework

It is often required to calculate the overlap matrix of a set of wavefunctions. This overlap matrix is symmetric and an optimal code will take advantage of this fact. Write a Fortran subroutine that calculates $\mathbf{B} \leftarrow \mathbf{A}^{\dagger}\mathbf{A}$ for real*8 matrices $\mathbf{A}$ of dimensions (m,n). A template of this BLAS-like routine is given below. The performance of the subroutine should be measured in Mflops for some values of n and m. The number of Mflops is calculated from m, n, and the cpu-time tcpu to perform the calculation (AIX Fortran routine mclock()).

$$Performance[Mflops] = m * n * (n+1)/tcpu * 10^{-6}$$

```
      subroutine olap(n,m,x,a)
c..calculate x=a(t)*a
      implicit real*8 (a-h,o-z)
      dimension x(n,*),a(m,*)


      return
      end
```