Fortran 2003

Bill Long, Cray Inc.

21-May-2004

Fortran 2003

- Specification for Fortran 2003 (f03) is finished
- Standard should be official in September
- 569 pages including Appendices and Index

- Document is available at j3-fortran.org
- Also available internally at Cray
- Working document number is 04-007
- formats: postscript, pdf, ASCII text

CUG 2004

Fortran 2003

- f03 is a major revision of f95
- Will replace f95 as the definition of “Fortran”

- Selected new features:
- C interoperability
- Procedure pointers
- Object Oriented Programming
- Allocatable components, dummy arguments
- Asynchronous and Stream I/O
- IEEE arithmetic support

CUG 2004

Organization

- New f03 features
- Implementation status
- Fortran 2009 proposals

CUG 2004

C Interoperability overview

- The most popular new feature
- Relating Fortran and C types
- Sharing global data between Fortran and C
- Fortran can call C functions
- C can call Fortran procedures

C Interoperability - types

- **KIND constants in new intrinsic module**

```
use,intrinsic :: iso_c_binding

integer(c_long)     :: long_int_var
real(c_double)      :: double_real_var
integer(c_int32_t)  :: pid

type,bind(c)           :: timeval
    integer(c_long)    :: tv_sec
    integer(c_long)    :: tv_usec
end type
```

C Interoperability - enum

- **Enumerators match types with C**

```
enum,bind(c)

  enumerator :: red=4, blue, yellow

end enum
```

C Interoperability - Data

- **Sharing data values between Fortran and C**

```
module global_data
  use,intrinsic :: iso_c_binding
  integer(c_int),bind(c,name="Mc") :: mc

  common /tblock/ t
  common /mblock/ r,s
  real(c_float) :: r,s,t
  bind(c) :: /tblock/
end module global_data

int Mc; struct{float r,s} mblock; float tblock;
```

Fortran calling C

- Binding gets name mangling correct
- Allows for arguments passed by VALUE

```
use,intrinsic :: iso_c_binding

interface
   function kill(pid,sig) bind(c) result(r)
      import c_int, c_int32_t
      integer(c_int)             :: r
      integer(c_int32_t),value :: pid
      integer(c_int),    value :: sig
   end function kill
end interface
```

CUG 2004

C calling Fortran

```
subroutine cb_revcompl(db,dbc,dblen,m) bind(c)
use,intrinsic :: iso_c_binding

   integer(c_long),intent(in) :: db(*)
   integer(c_long),intent(out):: dbc(*)
   integer(c_long),value      :: dblen
   integer(c_long),value      :: m

! Code here
end subroutine cb_revcompl
```

```
void cb_revcompl(long *db, long *dbc,
                 long dblen, long m);
```

CUG 2004

New C related intrinsics

```
type(c_ptr)  -  matches C pointer type

c_loc(fobj) returns pointer to Fortran object

c_f_pointer(cptr,fptr[,shape]) creates a
    Fortran pointer from a C pointer

c_associated(cptr1 [,cptr2]) is like associated
    except for type(c_ptr) arguments

c_funloc(fproc) creates a C pointer to the
    Fortran procedure fproc.

c_f_procpointer(cfun, fptr) makes f03 pointer
```

CUG 2004

Procedure pointers

- **The new procedure declaration statement can be used to declare a procedure pointer.**

```
abstract interface
     function rfun(x)
     real,intent(in) :: x
     real            :: rfun
end interface

procedure(rfun),pointer :: funp => null()

funp => gamma
```

Object Oriented Programming

- User defined types can be extended
- parent type is available as a component
- Type bound procedures are allowed
- Possible to override type bound procedures
- FINAL procedures are a special case
- PASS object dummy arguments
- Polymorphic dummy arguments, CLASS(*)
- select type construct
- Benefit: code reuse and collapse of multi-layer derived types

Allocatable Components

- Allocatable components of derived types
- Was a TR - implemented almost everywhere
- New implicit allocation on assignment

```
type foo
   real,allocatable :: grid(:,:)
end type foo

type(foo) :: old,new

new = old ! causes new%grid allocation
```

Allocatable Dummy args

- Dummy arguments can be allocatable
- Was a TR - implemented almost everywhere
- The objects survive the procedure return
- Allocatable function results are a special case

```
subroutine getdata (lun, array, nwords)
integer,intent(in)                :: lun
real,allocatable,intent(out)      :: array
integer,intent(out)               :: nwords
… ! compute or read in nwords
allocate(array(nwords))
…
```

CUG 2004

Asynchronous I/O

- Allows other work to overlap I/O operations
- Similar to old buffer in/buffer out statements

```
open(10,file='big.dat',asynchronous='yes')

read(10,100,asynchronous='yes',id=iw) BIG
… ! other work that does not involve BIG

wait(10,id=iw)

… ! now BIG can be used
```

Stream I/O

- File is treated as a sequence of bytes
- Alternative to sequential and direct

```
character :: c,d
open(10,file='bytes',access='stream',
        form='unformatted')

! read 34'th byte in the file

read(10,pos=34) c

! read 35'th byte in the file

read(10) d
```

CUG 2004

IEEE support

- Enable and disable interrupts
- Control rounding modes
- Intrinsics return representations of Inf, NaN
- Inquiry intrinsics like IEEE_IS_NAN(x)
- Named constants for exception flags
- Inquire about hardware support

Implementation

- **Fortran 2003 Implementation status for the Cray Fortran compiler for X1.**

- **Features already implemented**
- **Features planned for Cray Fortran 5.3**

CUG 2004

Implementation status - 1

- **f03 features in Cray Fortran 5.2**

```
mixed component accessibility
public entities of private type
keywords in structure constructors
allocatable components
allocatable dummy arguments
allocatable function results
auto allocation of allocatable components
auto allocation of allocatable arrays (-ew)
VOLATILE attribute
INTENT specification for pointer arguments
MIN and MAX intrinsics for character args
```

Implementation Status - 2

- **f03 features in Cray Fortran 5.2 - continued**

```
specified lower bounds in pointer assignment
parameters in complex constants
PROTECTED attribute
enumerators
FLUSH statement
named constants for key I/O units
carriage control characters removed
access to command line arguments
access to environment variables
255 continuation lines (unlimited)
optional KIND arguments in several intrinsics
```

CUG 2004

Implementation Status - 3

- **f03 features in Cray Fortran 5.2 - continued**

```
C interoperability - all except:
  comma before bind(c) for subprograms
  c_funloc()
  c_f_procptr()
```

CUG 2004

Implementation Status - 4

- **f03 features planned for ftn 5.3 (October, 2004)**

```
type specs in array constuctors
[ … ] syntax for array constructors
pointer rank remapping
asynchronous and stream I/O
comma instead of . in formatted numbers
access to I/O error messages
procedure declarations
procedure pointers
renaming user defined operators
63 characters in names
finish KIND arguments in intrinsics
generic form of SYSTEM_CLOCK
finish C interoperability
```

CUG 2004

The next Fortran version

- WG5/J3 attention is now on the next revision.
- Planned for 2008 or 2009. I like f09.

- One feature already set: Submodules

- Major Features being considered:
  - Co-Arrays
  - Typeless
  - Generic programming

Submodules

- TR already written - out for vote now.
- Effectively part of f03.

- Put module procedure interfaces in the main module along with public data
- Put implementations of the procedures in submodules.
- Submodules in separate files. Good for:
  - > dividing up work on big projects
  - > avoiding compilation cascades

CUG 2004

Co-Array concepts

- Parallel programming is pervasive in HPC.
- Co-Array model is SPMD with N identical images of a program cooperating.
- Co-Arrays provide a simple syntax for referencing the memory in another image.
- It is possible to implement Co-Arrays on top of MPI or some other communications scheme so performance is no worse than MPI.
- Performance can be MUCH better than MPI.
- Ease of programming is significantly higher.

Co-Array examples

```
real :: a(100)[*]
real,allocatable :: b(:)[:]
integer :: odd_team(4) = [1,3,5,7]

allocate (b(n)[*])

mype = this_image()

x = a(20)[1]
call sync_all()
if (any(odd_team == mype)) then
   call sync_team( odd_team )
end if
```

TYPELESS

- New data “type”
- Has kind and rank, but not traditional type
- Basically a known size block of bits
- BOZ constants are typeless
- List directed I/O is in Z format
- Sizes corresponding to the integers and reals

```
typeless(8) :: X, Array(10)

X = z'0000ffff0000ffff'
```

CUG 2004

TYPELESS operations

- Assignment involves no change in bits
- Argument association based on size only, type matching not involved.
- Bitwise operations (and, or, xor, not) defined
- Relational operations (< > == /= <= >=) defined
- Can ‘cast’ using REAL, INT, …
- Replaces many of the uses of unsigned ints.
- Standardizes many common intrinsics, including shiftr, shiftl, dshiftr, dshiftl, popcnt, leadz. Overloads old ones like IOR, IAND, IEOR.

CUG 2004

TYPELESS examples

```
real(8)      :: r,s(10)
integer(8)   :: i,j(10)

r = z'fff0000000000000' ! -Inf
call bcst(r,s,10)
i = typeless(r)
call bcst(i,j,10)

subroutine bcst(x,y,n)
  integer  :: n
  typeless(8) :: x,y(n)
     y = x
end subroutine bcst
```

Generic Programming

- Capability similar to C++ templates
- Avoids having many very similar procedures
- Actual versions of the routines created at compile time based on call usage
- There are several competing proposals still on the table.

Links

- **Bill Long - longb@cray.com**
- **J3 web site: j3-fortran.org**
- **General information: fortran . com**