Further Interoperability of Fortran with C

Dan Nagle

Welcome

Outline

- Existing Interoperability
- Why new features ? (Why TS 29113 ?)
- The Existing **iso_c_binding** Intrinsic Module
- Extensions to Fortran 2008
- The new **ISO_Fortran_binding.h** header file
  - constants (to set types, attributes, return codes)
  - types (to hold index values and descriptors)
  - functions (to create and manipulate descriptors)

Existing Interoperability

- The Common Subset Interoperates (Clause 15)
  - Kind type parameters from iso_c_binding (Table 15.2)
  - Derived types and procedure interfaces declared with bind(c)
  - Fortran scalars interoperate with C variables
    - Fortran value attribute where the formal parameter is a non-pointer C variable
    - Otherwise, the C formal parameter is a pointer
  - Fortran 77 arrays are passed as pointers
  - Anything requiring a descriptor is undefined

Why New Features?

- Fortran makes use of descriptors for pointers, allocatable variables, assumed-shape arrays
- C programmers want to be able to interact with these objects
- Leads to reverse-engineering of descriptors
  - Complicated #if cases on programmer's side
  - Constrains upgrades on compiler's side
- Latest (complete) draft of TS 29113 is N1885

The Existing iso_c_binding Intrinsic Module

- Kind type parameters for C types
  - See Table 15.2 in 10-007r1 (Fortran 2008)
- Constants for C control characters (\n, \t, and the rest) (Table 15.1)
- Fortran intrinsic module procedures:
  - to convert between C and Fortran pointers
  - to obtain C addresses: c_loc(), c_funloc()
  - to inquire: c_associated(), c_sizeof()

Extensions to Fortran 2008

- To interoperate with C void * formal parameters
  - For example, MPI choice buffers
  - Breaks the Fortran strong type system (TKR)
- To support asynchronous processing on the C side
  - Message passing, GPU programming
  - C11 threading ??

Assumed Type

```
! assumed type dummy argument
function f( b)
! declare b
type( *), intent( in) :: b
! can be called as
c = f( i)
! or as
c = f( x)
```

Assumed Type Restrictions

- Assumed type applies only to dummy arguments and requires an explicit interface
- An assumed type dummy argument can only be passed to another assumed type dummy argument, or to c_loc()
- Assumed type is otherwise treated as unlimited polymorphic (that is, class(*))
- There is no way in Fortran to query the type

Assumed Rank

```
! assumed rank dummy argument
function f( b)
! declare b
real, intent( in) :: b(..)
! can be called as
c = f( array1( :))
! or as
c = f( array3( :, :, :))
```

Assumed Rank Restrictions

- Assumed rank applies only to dummy arguments and requires an explicit interface
- An assumed rank dummy argument can only be passed to another assumed rank dummy argument, or to c_loc()
- Assumed rank dummy arguments may not have the value attribute, nor be a coarray
- Assumed rank dummy arguments may have the contiguous attribute

Combined

```
! assumed rank & assumed type
function f( b)
! declare b void *
type( *), intent( in) :: b(..)
! can be called as
c = f( i)
! or as
c = f( zarray3( :, :, :))
```

New & Improved Intrinsics

- shape(), size(), and ubound() are extended to handle objects of assumed rank
  - by returning a rank-1 array of variable size (when called without dim=)
- the new rank() intrinsic inquires of the rank of the object passed as an assumed rank dummy
  - When passed a scalar, a zero-sized array is the result

Fewer Restrictions on Dummy Arguments in Bind(C) Interfaces

- An argument in an interface with bind(c) may have the allocatable, pointer, or optional attribute, or be an assumed-shape array
- optional and value may not appear on the same dummy argument
- allocatable and pointer may not appear if the type has default initialization, type bound procedures, or a final procedure

The Asynchronous Attribute

- The asynchronous attribute existed to support Fortran asynchronous I/O
- Now, an asynchronous attribute may appear on a dummy argument in a bind(c) interface
- There must be a procedure to initialize communication and a procedure to complete the communication
  - But communication is not defined (simply, these procedures are processor dependent)

Combining All the Above

- There is now a completely standard means for a Fortran procedure to invoke asynchronous MPI calls, using the MPI choice buffer (that is, a formal parameter that is void *)
- A single Fortran procedure with one interface may handle all types and ranks
  - For example, MPI send and receive

ISO_Fortran_binding.h

- Constants to specify Fortran attributes
  - pointer, allocatable, assumed-shape
  - See Table 8.1 in N1885
- Types for descriptors, array extents, bounds
- Macros / Functions to create, query, and update descriptors
- Inquiry of Fortran attributes
  - contiguous
- Return codes

CFI_index_t

- A typedef for a signed integer type that can hold the result of subtracting two pointers
- It holds an index value
- It is used in constructing other types

CFI_dim_t

- Represents one dimension of a Fortran array
- It is a named struct
  - CFI_index_t lower_bound; // of the dimension
  - CFI_index_t extent; // count of elements
  - CFI_index_t sm; // stride in bytes

CFI_rank_t

- A typedef for a standard integer type
- It holds the largest rank supported

CFI_type_t

- A typedef for a standard integer type
- It holds any type specifier
  - See Table 8.2 in N1885

CFI_attribute_t

- A typedef for a standard integer type
- It holds any attribute code
  - See Table 8.1 in N1885

CFI_cdesc_t

- A named struct named by a typedef
  - contains a flexible array member
- The type of a Fortran descriptor for use in C
  - void * base_addr; size_t elem_len; int version;
  - CFI_rank_t rank; CFI_type_t type;
  - CFI_attribute_t attribute;
  - CFI_dim_t dim[];
- Order requirements on sm sizes to prevent overlap

CFI_CDESC_T(int)

- The int specifies the rank of the Fortran object
- The macro expands to the name of a type suitable for defining a descriptor of the rank specified
  - a pointer to a variable so declared may be cast to a CFI_cdesc_t *

CFI_MAX_RANK

- A macro expanding to a standard integer type to hold the largest rank supported
  - the value must be >= 15

CFI_VERSION

- A macro expanding to an integer value that is used to track when incompatible changes are made to the definitions in the header file

Attribute Codes

- CFI_attribute_assumed
  - Fortran assumed-shape array
- CFI_attribute_allocatable
  - Fortran allocatable variable
- CFI_attribute_pointer
  - Fortran pointer variable
- CFI_attribute_unknown_size
  - Fortran assumed-size array

Type Codes

- See Table 8.2 of N1885
- Type codes for all the usual standard C types
  - including size_t, _Bool, intmax_t, ptrdiff_t, void *, pointer to function, structs
  - other (when type is unspecified)

Error Codes

- See Table 8.3 in N1885
- CFI_SUCCESS is 0
- Other error codes are distinct from any other constant in the header
- A processor may detect other errors
- When several errors occur, which error is reported is processor-dependent

CFI_address()

- Computes the address of an array element
- Returns a void *
- Takes a CFI_cdesc_t *
- Takes a CFI_index_t []

CFI_establish()

- Initializes a CFI_cdesc_t object
- Completes
  - base_addr // if appropriate, or NULL
  - attributes
  - type
  - elem_len // if appropriate, or NULL
  - rank
  - extents // if appropriate, or NULL

CFI_allocate() / CFI_deallocate()

- Allocate or deallocate a Fortran allocatable object by the same mechanism used by the Fortran compiler
  - so it duplicates Fortran allocate and deallocate statements

CFI_is_contiguous()

- Returns 1 if the object is contiguous (has the Fortran contiguous attribute) and 0 otherwise

CFI_section()

- Returns a CFI_cdesc_t object describing an array section
- For example, given a CFI_cdesc_t for a rank-1 array **a**, returns a CFI_cdesc_t for **a(f: l: i)**
  - applies to arrays of any rank
  - the rank is specified in the CFI_cdesc_t object

CFI_select_part()

- Given an array of derived type, returns an array of part of it
- For example, given

  type, bind(c) :: foo_t

  type(goo_t) :: c

  end type

  – and a **CFI_cdesc_t** for type(foo_t) :: **a(n)**

  – return a **CFI_cdesc_t** for **a(:)% c**

CFI_setpointer()

- Associates a Fortran pointer with a target
- For example, given a CFI_cdesc_t describing
  - type(x), pointer :: px
- And a CFI_cdesc_t describing
  - type(x), target :: tx
- Updates the **CFI_cdesc_t** for px as if the pointer assignment **px => tx** in Fortran

Restrictions

- Mind the Pointer Lifetimes (of course)
- >>> Don't try to do something with Fortran objects in C if it can't be done in Fortran <<<
- Don't use names beginning with CFI_ in any file where the header is included

Further Interoperability of Fortran with C

Dan Nagle

Thank you