What’s New in Intel® Fortran 16.0

Part of Intel® Parallel Studio XE 2016

May 13, 2015
New and Changed Features in Fortran 16.0

• Features from Fortran 2008
• Further C Interoperability from Fortran 2015
• OpenMP 4.1 TARGET NOWAIT | DEPEND
• !DEC$ BLOCK_LOOP directive
• -init enhancements
• -fpp-name option
• VS2013 Shell
Features from Fortran 2008
Submodules (Fortran 2008)

The Problem:

• Any edit to a module, no matter how trivial, requires recompilation of all sources that USE that module, directly or indirectly

• Can cause a “recompilation cascade” in builds, greatly lengthening build time

The Solution:

• Submodules separate interface from implementation

• Changes in a submodule don’t force recompile of module or sources that use the module (unless interfaces change)
Submodules (Fortran 2008)

- Interface goes in module
- Use MODULE SUBROUTINE or MODULE FUNCTION in interface body
- Submodule in separate source
  - SUBMODULE(parent-id) submodule-name
  - parent-id is ancestor-module-name [: parent-submodule-name]
- Can have tree structure of submodules
- Declarations in submodule are visible only to submodule (and children)
Submodules (Fortran 2008)

• Implementation inside submodule can either repeat the interface from the module (with the MODULE prefix) or use MODULE PROCEDURE
  • Repeating interface puts procedure declaration next to its implementation, but duplicates text
  • If repeated, names, types and characteristics of dummy arguments must match
  • MODULE PROCEDURE means writing out the interface only once
• Submodules can enable circular reference: module A can reference module B, submodule of B can use module A
Submodules (Fortran 2008)

module bigmod
  ...
  interface
  module subroutine sub1(arg)
  integer arg
  ...
  module function func2(arg)
  real arg
  ...
  module subroutine sub47(arg)
  logical arg
  ...
  end interface
end module bigmod

submodule (bigmod) bigmod_submod
  contains
  module subroutine sub1(arg)
  integer arg
  ...
  module function func2(arg)
  real arg
  ...
  module procedure sub47
  ...
end submodule bigmod_submod

Changes in the submodule don’t force recompilation of uses of the module – as long as the interface doesn’t change
IMPURE ELEMENTAL (Fortran 2008)

- In Fortran 2003, ELEMENTAL procedures are PURE
  - No I/O, no side-effects, can call only other PURE procedures
- New IMPURE prefix allows non-PURE elemental procedures
  - Can do I/O, call RANDOM NUMBER, etc.
EXIT from BLOCK (Fortran 2008)

• When we first implemented BLOCK in Fortran 15, we didn’t support EXIT from a BLOCK – now we do

• EXIT from other named constructs still in the future

outer: block
  do i = 1, num_in_set
  if ( x == a(i) ) exit outer
  end do
  call r
end block outer
Features from Fortran 2015
Further C Interoperability (Fortran 2015)

TS29113 on “Further Interoperability of Fortran with C” to be part of Fortran 2015. Motivations include:

- Support the needs of MPI3
- Provide Fortran equivalent of C’s “void*” – assumed type and rank
- Enable C code to see array bounds, manipulate pointers and allocatables
- Extend interoperable interfaces to ALLOCATABLE, POINTER, assumed shape, CHARACTER(*) - all passed by new “C Descriptor”
- OPTIONAL allowed in interoperable interface
- Extend ASYNCHRONOUS beyond I/O
- Relax restrictions
Assumed Type

- New declaration-type-spec syntax TYPE( *)
- **Dummy argument only**
- Unlimited polymorphic – no declared type
  - Effectively C’s void* - no type information passed
  - Must use C_LOC/C_F_POINTER to cast to Fortran type
  - Can't use SELECT TYPE
- May not have the ALLOCATABLE, CODIMENSION, INTENT(OUT), POINTER, or VALUE attribute
- Must also be assumed-shape or rank to pass to assumed-rank dummy
Assumed Rank

- New *array-spec* syntax `DIMENSION( .. )`
- **Dummy argument only**
- Rank assumed from effective argument, can be scalar
- May not have `CODIMENSION` or `VALUE` attribute
- Type/Kind/Rank matching extended so that assumed-rank matches any rank
- Use `C_LOC/C_F_POINTER` to cast to specific rank
- New `RANK` intrinsic returns rank of argument (could be 0 => scalar)
- Pass by descriptor (C descriptor if `BIND(C)`)

---

Optimization Notice

Copyright © 2015, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.*
Relaxed argument attribute restrictions

- Dummy arguments to interoperable procedures may now have ALLOCATABLE, OPTIONAL, or POINTER attribute

- OPTIONAL and VALUE can’t be combined

- Default initialization not allowed for ALLOCATABLE or POINTER

- Assumed-shape arguments now allowed, passed by C descriptor

- Character assumed-length (CHARACTER(*)) arguments now allowed, passed by C descriptor! Potential trouble spot for programmers.

- If ALLOCATABLE passed to C routine with INTENT(OUT) interface, gets deallocated before call
Miscellaneous dummy argument changes

- Assumed-rank requires explicit interface
- Rank-zero actual arguments have a shape that is zero-sized
- Assumed-rank dummy can be associated with assumed-size actual
  - SHAPE, SIZE, UBOUND defined for this case
  - Last element of SHAPE is -1
  - SIZE of the last dimension is -1
  - SIZE of the whole array is -1 times size of the other dimensions
  - UBOUND of last dimension is LBOUND-2
C Descriptors

Goals:

• Provide a standard way to pass type and shape information to and from C
• Allow C code to allocate and deallocate ALLOCATABLE and POINTER variables
• Provide a way for C code to create array sections and to do pointer assignment

ISO_Fortran_binding.h and a support library usable by C provided by Fortran
• Typedefs, macros, functions for manipulating C descriptors
Fortran-C Exchange of C Descriptors

• For BIND(C) routines, where standard says that argument is passed by C descriptor, Fortran compiler automatically creates C descriptor, passes it to C routine, then reflects changes on return

• Fortran routines with BIND(C) automatically convert C descriptor to internal representation (if appropriate) on entry, convert back on exit

• Fortran code never sees C descriptors
ISO_Fortran_binding.h

- Header file is in Intel Fortran Include folder; functions included in Fortran compiler support library (libifcore)
- Definitions of structures CFI_cdesc_t and CFI_dim_t
- Typedefs for CFI_attributes_t, CFI_index_t, CFI_rank_t, CFI_type_t
- Macro CFI_CDESC_T for local declaration of descriptor
- Macros expanding to integer constants
- Functions for manipulating C descriptors
- All names start with CFI_
CFI_cdesc_t

typedef structure with at least the following members:

- `void * base_addr` – Null pointer if disassociated, processor-dependent if zero-sized (1st in structure)
- `size_t elem_len` – Storage size in bytes of scalar or single element (2nd)
- `int version` – processor-dependent version number (3rd)
- `CFI_rank_t` – rank of array, 0 if scalar
- `CFI_type_t` – specifier of type of object
- `CFI_attribute_t` – designates allocatable/pointer/other
- `CFI_dim_t` – dim[] – array of bounds and stride information (last)
Macros and typedefs

- CFI_CDESC_T(n) macro declares memory for descriptor of rank n
- CFI_index_t - typedef name for signed integer type that can represent the result of subtracting two pointers
- CFI_MAX_RANK – largest rank supported (no less than 15)
- CFI_rank_t, CFI_type_t, CFI_attribute_t – typedefs for integer type that can hold rank/type/attribute code
- CFI_VERSION – processor-dependent macro encoding this version of ISO_Fortran_binding.h
Functions

- CFI_address – compute the C address of an object
- CFI_allocate – allocate memory for an object
- CFI_deallocate – deallocate memory for an object
- CFI_establish – fill in a C descriptor
- CFI_is_contiguous – test contiguity of an array
- CFI_section – update a descriptor to describe an array section
- CFI_select_part – update a descriptor to describe a part (component, etc) of an array
- CFI_setpointer – do pointer assignment
Restrictions for Programmers on C descriptors

- Shall not be initialized, updated, nor copied other than by calling the functions in ISO_Fortran_binding.h
- Shall not be modified if Fortran interface says INTENT(IN) or if descriptor is for nonpointer, nonallocatable object
- Base address shall not be accessed before it is given a value if corresponding Fortran dummy argument has POINTER and INTENT(OUT)
- Allocatable/Pointer objects shall be allocated/deallocated only by calls to CFI_allocate and CFI_deallocate
Example

use, intrinsic :: iso_c_binding
interface
  function c_alloc (array) bind(C)
    import
    integer(C_INT) :: c_alloc
    real(C_FLOAT), intent(out), allocatable, dimension(:) :: array
  end function c_alloc
end interface
real(C_FLOAT), allocatable, dimension(:) :: my_array
if (c_alloc(my_array) == 0) then
  print *, lbound(my_array), ubound(my_array); print *, my_array
end if
end

#include "ISO_Fortran_binding.h"
extern int c_alloc (CFI_cdesc_t * descr) {
  int ret, i; float * array;
  CFI_index_t lower = 0, upper = 10;
  ret = CFI_allocate (descr, &lower, &upper, 0); // No elem_len
  if (ret == CFI_SUCCESS) {
    array = descr->base_addr;
    for (i=lower;i<=upper;i++) {array[i] = (float) i;}
  }
  return ret;
}
ASYNCHRONOUS communication

• ASYNCHRONOUS attribute extended to apply to variables used for asynchronous communication
• Similar effect to existing I/O-related ASYNCHRONOUS
• Added to benefit MPI
• Definition change only, compiler already treats variables appropriately
Other Features
Other New Features

• OpenMP 4.1
  • !OMP$ TARGET NOWAIT – current task may continue execution without waiting for the target to finish
  • !OMP$ TARGET DEPEND – treated as if DEPEND had been specified for implicit TASK construct enclosing TARGET
• !DIR$ [NO]BLOCK LOOP enables or disables loop blocking for following loop
  • -init now applies to locals, allocatables, pointer allocations
  • -fpp-name lets you supply your own fpp preprocessor
• VS2013 Shell replaces VS2010 Shell on Windows
Legal Disclaimer & Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED "AS IS". NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Copyright © 2015, Intel Corporation. All rights reserved. Intel, Pentium, Xeon, Xeon Phi, Core, VTune, Cilk, and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804