Using Fortran 2003 features in OpenMP programs

Kelvin Li

kli@ca.ibm.com

IBM Canada Lab

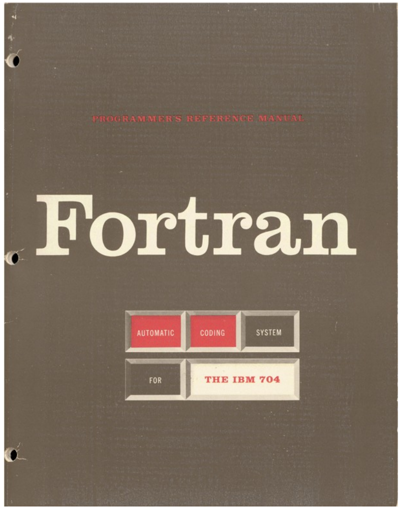
OpenMPCon, Oct 3-5, 2016 | Nara, Japan

Outline

- Fortran 2003 standard
- Fortran 2003 features in OpenMP spec
- Remaining works
- Summary

Overview of Fortran 2003

- formally known as ISO/IEC 1539-1:2004
- published in 2005
- a major update from the previous revision (Fortran 95)
- added many modern programming features



PROGRAMMER'S REFERENCE MANUAL

Fortran

AUTOMATIC CODING SYSTEM

FOR THE IBM 704

Overview of Fortran 2003

Data enhancements and object orientation

- IEEE module
- parametrized derived type
- procedure pointers
- type extension
- finalization
- polymorphic entities
- ASSOCIATE construct
- allocatable scalars
- allocatable character length
- SELECT TYPE construct
- Enumerations

Overview of Fortran 2003

Miscellaneous enhancements

- structure constructors
- allocate statement
- assignment to an allocatable array
- transferring an allocation
- more control of access from a module
- renaming operators on the USE statement
- pointer assignment
- pointer intent
- the VOLATILE attribute
- the IMPORT attribute
- intrinsic modules
- access to the computing environment
- support for international character sets
- lengths of names and statements
- binary, octal, and hex constants
- array constructor syntax
- specification and initialization expressions
- complex constants
- changes to intrinsic functions

Overview of Fortran 2003

Input/output enhancements

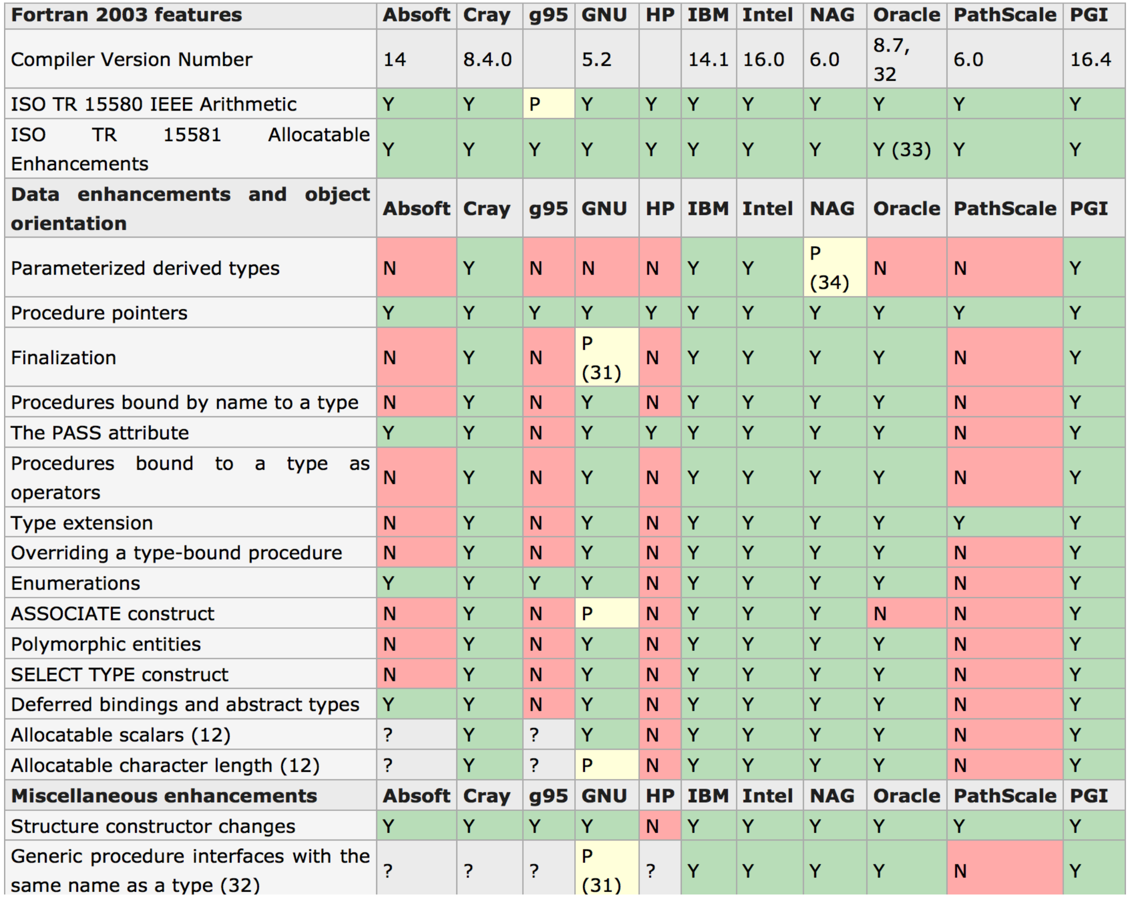
- derived type input/output
- asynchronous input/output
- FLUSH statement
- IOMSG= specifier
- stream access input/output
- ROUND= specifier
- DECIMAL= specifier
- SIGN= specifier
- kind type parameters of integer specifiers
- recursive input/output
- intrinsic function for newlines characters
- input and output of IEEE exceptional values
- comma after a P edit descriptor

Overview of Fortran 2003

Interoperability with C

- interoperability of intrinsic types
- interoperability with C pointers
- interoperability of derived types
- interoperability of variables
- interoperability of procedures
- interoperability of global data

Overview of Fortran 2003



Fortran 2003 features	Absoft	Cray	g95	GNU	HP	IBM	Intel	NAG	Oracle	PathScale	PGI
Compiler Version Number	14	8.4.0		5.2		14.1	16.0	6.0	8.7, 32	6.0	16.4
ISO TR 15580 IEEE Arithmetic	Y	Y	P	Y	Y	Y	Y	Y	Y	Y	Y
ISO TR 15581 Allocatable Enhancements	Y	Y	Y	Y	Y	Y	Y	Y	Y (33)	Y	Y
Data enhancements and object orientation	**Absoft**	**Cray**	**g95**	**GNU**	**HP**	**IBM**	**Intel**	**NAG**	**Oracle**	**PathScale**	**PGI**
Parameterized derived types	N	Y	N	N	N	Y	Y	P (34)	N	N	Y
Procedure pointers	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Finalization	N	Y	N	P (31)	N	Y	Y	Y	Y	N	Y
Procedures bound by name to a type	N	Y	N	Y	N	Y	Y	Y	Y	N	Y
The PASS attribute	Y	Y	N	Y	Y	Y	Y	Y	Y	N	Y
Procedures bound to a type as operators	N	Y	N	Y	N	Y	Y	Y	Y	N	Y
Type extension	N	Y	N	Y	N	Y	Y	Y	Y	Y	Y
Overriding a type-bound procedure	N	Y	N	Y	N	Y	Y	Y	Y	N	Y
Enumerations	Y	Y	Y	Y	N	Y	Y	Y	Y	N	Y
ASSOCIATE construct	N	Y	N	P	N	Y	Y	Y	N	N	Y
Polymorphic entities	N	Y	N	Y	N	Y	Y	Y	Y	N	Y
SELECT TYPE construct	N	Y	N	Y	N	Y	Y	Y	Y	N	Y
Deferred bindings and abstract types	Y	Y	N	Y	N	Y	Y	Y	Y	N	Y
Allocatable scalars (12)	?	Y	?	Y	N	Y	Y	Y	Y	N	Y
Allocatable character length (12)	?	Y	?	P	N	Y	Y	Y	Y	N	Y
Miscellaneous enhancements	**Absoft**	**Cray**	**g95**	**GNU**	**HP**	**IBM**	**Intel**	**NAG**	**Oracle**	**PathScale**	**PGI**
Structure constructor changes	Y	Y	Y	Y	N	Y	Y	Y	Y	Y	Y
Generic procedure interfaces with the same name as a type (32)	?	?	?	P (31)	?	Y	Y	Y	Y	N	Y

- implementing the full standard still work in progress
- cost? interest?

http://fortranwiki.org/fortran/show/Fortran+2003+status

Rebasing to Fortran 2003

- fortunately, many new features do not impact the behavior with OpenMP
- for example
  - input/output enhancements
  - the new `ISO_FORTRAN_ENV` module
  - syntactic enhancements
    - array constructor syntax: `[1,2,3,4]` as an alternative for `(/1,2,3,4/)`
    - complex constants: allow named constants in a complex constant

      ```
      real, parameter :: one=1.0, zero=0.0
      complex :: c
      c = (one, zero)
      ```

Rebasing to Fortran 2003

- the current base language is FORTRAN 77, Fortran 90 and Fortran 95
- began the effort to rebase to Fortran 2003 in V4.0
- investigated the list of new features in terms of the impact to the spec
- areas of investigation
  - how the new objects in Fortran 2003 interact with the data-sharing attribute clauses?
  - how the new language constructs interact with the OpenMP constructs etc?
  - for post V4.0, we also need to care about the interaction with the data-mapping attribute clauses?

Allocatable entities

- Fortran 2003 has major enhancement on the allocatable entities

```
subroutine sub(y)
  real, allocatable :: arr(:) ! F90: allocatable array
  real, allocatable :: y      ! F2003: dummy argument
  real, allocatable :: x      ! F2003: allocatable scalar
  type dt
    real, allocatable :: z    ! F2003: allocatable component
  end type

  x = 2.0                     ! F2003: allocate and then assign
  allocate(arr(10))
  arr = [ 1.0, 2.0 ]          ! deallocate, allocate and then assign
end subroutine

real :: t(15)
call sub(t)                   ! dummy argument becomes allocated
```

Allocatable entities

```
subroutine foo(k, n)
  integer, allocatable :: x(:)
  integer, intent(in) :: n, k
!$omp parallel private(x)
  x = [ (k,i=1,n) ]
  ...
!$omp end parallel
end subroutine
```

```
subroutine foo(k, n)
  integer, allocatable :: x(:)
  integer, intent(in) :: n, k
  allocate(x(n))
!$omp parallel private(x)
  x = [(k,i=1,n)]
  ...
!$omp end parallel
  deallocate(x)
end subroutine
```

- the latest spec supports the Fortran 2003 allocatable enhancements
- using some of the features may cause performance impact
- both are legal OpenMP code
- however, taking the advantage of re-allocation may incur some compiler / runtime overhead
- each private copy is not allocated when entering the parallel region
  - allocation is done on each thread
- each private copy is allocated when entering the parallel region
  - no allocation is needed as long as the same shape of array is assigned to it
- avoid to have explicit allocation/re-allocation inside a parallel region if feasible

Allocatable entities

```
  real, allocatable :: x(:)
  real :: y(12)
  allocate(x(5))

!$omp parallel sections lastprivate(x)
!$omp section
  x = [ 2.1, 3.1, 10.7, 1.4, 13.2 ]
  ... = x
!$omp section
  x = y
  ... = x     ! size(x) is 12
!$omp end parallel sections
! size(x) == size(y)
```

- the semantic of the intrinsic assignment is enhanced in the base language
- the spec defines the lastprivate semantic as if by intrinsic assignment
- the implication of having allocatable arrays on the lastprivate clause is that the size and shape may be changed due to the intrinsic assignment

Associate construct

- *“associates name entities with expression or variables during the execution of its block”*

```
c = -1
associate (as => c)
  print *, as          ! -1
  as = 10
  print *, c, as       ! 10 10
end associate
  print *, c            ! 10
```

```
x = 3 ; y = 4
associate (Pyth => sqrt(x*x + y*y))
  x = 5
  y = 12

  print *, Pyth, sqrt(x*x + y*y) ! 5 13
end associate
```

- the feature provides a convenient way to replace some complex expression (e.g. `a(j)%b(lbound(x,1):ubound(x,1),lbound(y,1):ubound(y,1))%c`) by a simple name in a block
- the *associate-name* (`as` and `Pyth`) is not visible outside the associate construct
- the *selector* (c) is visible inside the associate construct

Associate construct

- the special characteristics of the *associate-name* and the *selector* introduces some difficulties in working out the interaction with the data-sharing attribute clauses
- having the *associate-name* on the data-sharing attribute clauses does not quite make sense
    - the *associate-name* does not have its own storage

```
associate (Pyth => sqrt(x*x + y*y))
  ...
!$omp parallel private(Pyth)  ! invalid
  Pyth = ...
!$omp end parallel
  ...
end associate
```

```
associate (b => a)
  ...
!$omp parallel private(b)  ! invalid
  b = ...
!$omp end parallel
  ...
end associate
```

Associate construct

- using the associate construct inside an OpenMP construct is easier to handle
- the *associate-name* inherits the data-sharing attribute from the *selector*
  - the *associate-name* is associated with the private copy
- caution must be taken when using this feature
  - easy to use the associate construct to access the original list item of the private variable

```
  ...
!$omp parallel private(nthread)
  nthread = omp_get_thread_num()
  associate (p => nthread)
    ... = p  ! p is private
  end associate
!$omp end parallel
  ...
```

```
  x = ...
 associate (assoc_name => x)

!$omp parallel private(x)
  x = omp_get_thread_num()
  ... = assoc_name  ! original list item
!$omp end parallel

 end associate
```

Other major enhancements

- the user defined reduction also introduces complication in describing the reduction-identifier being defined in a module
- the *reduction-identifier* is not a Fortran entity, the USE mechanism cannot apply directly
- rules are set up to set the behavior

```
module m
  interface operator(.add.)
    module procedure add_t
  end interface
!$omp declare reduction(.add.: dt: add_dt(omp_in, omp_out))
end module

  use m
  type(dt) :: xdt
!$omp parallel do reduction(.add.: xdt)
  do i=1, N
    xdt = ...
  enddo
```

Remaining work

- remaining F2003 features
- polymorphic entities
- parametrized derived types
- complexity in Fortran by-descriptor objects (i.e. allocatable variable or pointer) with the mapping mechanism as well as the declare target mechanism

Summary

- works done for rebasing to Fortran 2003
- cautious in using the new features
- more work to be done

Thank
You