Introduction

• ‘Fortran 2018’ is the current Fortran standard (it was previously called Fortran 2015)
• This talk will introduce some of the new features
• We will specifically consider the status wrt. the Cray Fortran compiler
Some History

• ((really) old stuff we don’t care about)

• FORTRAN 77
• Fortran 90
• Fortran 95
• Fortran 2003
• Fortran 2008
• Fortran (I mean 2018 or really ISO/IEC 1539-1:2018)
Some History

• **1954** ((really) old stuff we don’t care about)

• **1978** FORTRAN 77

• **1991** Fortran 90

• **1997** Fortran 95

• **2004** Fortran 2003

• **2010** Fortran 2008

• **2018** Fortran (I mean 2018 or really ISO/IEC 1539-1:2018)
Cray Implementations

- **1954** (really old stuff we don’t care about)
- **1978** FORTRAN 77
- **1991** Fortran 90
- **1997** Fortran 95
- **2004** Fortran 2003
  - **2008** CCE 7.0
- **2010** Fortran 2008
  - **2012** CCE 8.1
- **2018** Fortran (I mean 2018 or really ISO/IEC 1539-1:2018)
  - Targeting CCE 10.0 in mid 2020
Fortran 2018

• A minor Revision
• Incorporates
  • TS29113:2012 Further Interoperability of Fortran and C
  • TS18508:2015 Additional Parallel Features in Fortran
  • Many other features
  • Some of these features already supported by compilers
• For more details, see John Reid's "The New Features of Fortran 2018", WG5 N2161:
  https://isotc.iso.org/livelink/livelink?func=ll&objId=19867230&objAction=Open
**Fortran 2018 Compatibility** - No known issues,

THE FOLLOWING FEATURES ARE DEFERRED AND NOT YET IMPLEMENTED

- **TEAMS**: TEAM_TYPE, FORM TEAM, CHANGE TEAM, SYNC TEAM, GET_TEAM, TEAM_NUMBER; TEAM and TEAM_NUMBER arguments for the IMAGE_INDEX and NUM_IMAGES intrinsics, TEAM argument for the THIS_IMAGE intrinsic

- **CO_BROADCAST**, **CO_REDUCE**, **CO_SUM**, **CO_MIN** and **CO_MAX** collectives (*partial support - details below*)

- Optional integer type-spec in implied-do in DATA and array constructor

- Locality clauses for DO CONCURRENT

- Support for IEEE 60559:2011 changes (*partial support - details below*)

- Extended syntax for image selectors

- Failed images support: FAIL IMAGE statement, FAILED_IMAGES, STOPPED_IMAGES, and IMAGE_STATUS intrinsics; LOCK modifications to allow for failed images; CRITICAL construct to handle failed images; Modifications to STAT= specifier values in image control statements for failed images; STAT and ERRMSG arguments for the MOVE_ALLOC intrinsic and modified semantics for failed images

Further Interoperability of Fortran with C

• Feature has matured since introduction in Fortran 2003 up to the TS in 2012.
• Many changes motivated by desire for proper interface to MPI
• Fortran 2003 provided for the ISO_C_BINDING module and
  • Interoperable types
  • Interoperability of procedures/functions
  • Interoperability for global data (eg. within modules, common)
  • Pass by value (VALUE attribute for dummy types)
  • Association of Fortran pointer with C pointer target (data and procedures)
• Fortran 2008 added C_sizeof()
Further Interoperability of Fortran with C…

• Significant new support from TS and Fortran 2018 features
• `CFI_cdesc_t` descriptor to describe a Fortran object passed into C
  • Captures type, element size, shape, data layout (dims,strides)
• Ability to create (and allocate) this descriptor from C via provided interfaces
• Assumed rank dummy arguments (actual can be scalar(rank 0))
  Can pass this on or test with RANK() or control flow with SELECT RANK
• Assumed TYPE dummies (match void * or a C descriptor)
• Support for optional arguments (non-present map to null pointer in C)
• Fortran subscripting support from C (to obtain element location)
• ASYNCHRONOUS attribute can be used to indicate variables can change by
  means other than Fortran. Can apply the attribute within a BLOCK construct.
We can pass any rank and type

Example: Assumed rank and type

```fortran
module interoperate

interface
    subroutine describe(a) bind(c,name='describe')
    use, intrinsic :: iso_c_binding
    type(*), dimension(..), intent(in) :: a
    end subroutine describe
end interface

end module interoperate

program ex
use interoperate
implicit none
real r(3,100,100)

call describe(r(1,:,:))

end program ex
```
#include "ISO_Fortran_binding.h"
void describe(CFI_cdesc_t *desc){
    CFI_rank_t rank,i;
    CFI_type_t type;
    CFI_attribute_t attribute;
    CFI_dim_t *dim;
    char stype[26],sattr[26];
    rank = desc->rank;
    dim = desc->dim;
    type = desc->type;
    printf("%s object\n Rank: %d\n shape: (", attr_tostr(sattr,desc->attribute,26),rank);
    for (i=0;i<rank;i++){
        if (dim[i].extent == (-1)) {
            printf("%d:*",(int)dim[i].lower_bound);
        } else {
            printf("%d:%d",(int)dim[i].lower_bound,
                    (int)dim[i].lower_bound+dim[i].extent-1);
        }
        if (i<rank-1) printf(",");
    }
    printf("\n type: %s\n",type_tostr(stype,type,25));
    printf(" elem_len: %lld\n",desc->elem_len);
    printf(" base address: %p\n",desc->base_addr);
}
Example: SELECT RANK

```fortran
subroutine set_identity(a) ! A should be 'square'
    integer, dimension(..), intent (inout) :: a

    ! a = 0 ! not legal here
    select rank(a)
        rank(2)
        a = 0
        do i=1:size(a,1); a(i,i) =1; end do
        rank(3)
        a = 0
        do i=1:size(a,1); a(i,i,i)=1; end do
        rank default
        stop 'set_identity: invalid rank'
    end select
end subroutine set_identity
```
Example: ASYNCHRONOUS attribute

! buf previously declared

BLOCK
  asynchronous :: buf

  ! Get new data for buf from another rank
  MPI_Irecv(buf,n,dt,src,tag,comm,request)

  call compute(a)   ! buf could change during this

  MPI_Wait(request,MPI_STATUS_IGNORE)

END BLOCK

edge = buf      ! Safe from changes here
Coarray-related updates - Events

- Events fully supported: EVENT POST, EVENT WAIT statements, EVENT_QUERY intrinsic subroutine, EVENT_TYPE type.
- Atomic updates of event variables (coarrays).
- This is an important change as it provides a non-collective split-phase synchronisation capability.
- Advantage over SYNC IMAGES – “A coarray that is of type EVENT_TYPE may be referenced or defined during execution of a segment that is unordered relative to the execution of another segment in which that coarray is defined.”
Relaxation code – SYNC ALL

```fortran
do iter=1,niter
  if (imgpos(1) .ne. 1) oldpic(0,:) = oldpic(size1, :) [imgpos(1)-1, imgpos(2)]
  if (imgpos(1) .ne. nimgs1) oldpic(size1+1,:) = oldpic(1,:) [imgpos(1)+1, imgpos(2)]
  if (imgpos(2) .ne. 1) oldpic(:,0) = oldpic(:, size2) [imgpos(1), imgpos(2)-1]
  if (imgpos(2) .ne. nimgs2) oldpic(:,size2+1) = oldpic(:,1) [imgpos(1), imgpos(2)+1]

  sync all ! Segment boundary

  do j = 1, size2
    do i = 1, size1
      pic(i,j) = 0.25*(oldpic(i-1,j)+oldpic(i+1,j)+oldpic(i,j-1)+oldpic(i,j+1)-edge(i,j))
    end do
  end do

  oldpic = pic
  sync all ! Segment boundary
end do
```
Relaxation code – EVENTS (1)

type( event_type ), allocatable :: ready(:) [:,:]
allocate( ready( 4 ) [nimgs1,*] )
do iter=1, niter
    if ( imgpos(1) .ne. 1 ) &
        event post( ready(1) [imgpos(1)-1, imgpos(2) ], stat=ierr )
    if ( imgpos(1) .ne. nimgs1 ) &
        event post( ready(2) [imgpos(1)+1, imgpos(2) ], stat=ierr )
    if ( imgpos(2) .ne. 1 ) &
        event post( ready(3) [imgpos(1), imgpos(2)-1], stat=ierr )
    if ( imgpos(2) .ne. nimgs2 ) &
        event post( ready(4) [imgpos(1), imgpos(2)+1], stat=ierr )
if ( imgpos(1) .ne. 1 ) then
  event wait( ready(2), stat=ierr )
  oldpic(0,:) = oldpic(size1, :)[imgpos(1)-1,imgpos(2)]
end if
if (imgpos(1) .ne. nimgs1) then
  event wait( ready(1), stat=ierr )
  oldpic(size1+1,:) = oldpic(1,:)[imgpos(1)+1,imgpos(2)]
end if
if (imgpos(2) .ne. 1) then
  event wait( ready(4), stat=ierr )
  oldpic(:,0) = oldpic(:, size2)[imgpos(1),imgpos(2)-1]
end if
if (imgpos(2) .ne. nimgs2) then
  event wait( ready(3), stat=ierr )
  oldpic(:, size2+1) = oldpic(:,1)[imgpos(1),imgpos(2)+1]
end if
Relaxation code – EVENTS (3)

do j = 1, size2
  do i = 1, size1
    pic(i,j) = 0.25 * &
    (oldpic(i-1,j) + oldpic(i+1,j) + &
     oldpic(i,j-1) + oldpic(i,j+1) - &
     edge(i,j) )
  end do
end do
oldpic = pic
end do

• Main calc loopnest
• SYNC ALL is gone completely (could’ve done with SYNC IMAGES instead)
• UNTIL_COUNT in EVENT WAIT is supported. Not useful in this example.
• Intrinsic subroutine EVENT_QUERY is supported. Not useful in this example.
Relaxation code – performance

niter = 1000000 (1M loop iterations)
Broadwell 2.6 GHz, 14-core, dual socket, HT on

<table>
<thead>
<tr>
<th>Code</th>
<th>Imgs</th>
<th>layout</th>
<th>looptime</th>
<th>diff</th>
</tr>
</thead>
<tbody>
<tr>
<td>sync all</td>
<td>4</td>
<td>2x2</td>
<td>35.95</td>
<td></td>
</tr>
<tr>
<td>Events</td>
<td>4</td>
<td>2x2</td>
<td>16.66</td>
<td>x2 faster</td>
</tr>
<tr>
<td>sync all</td>
<td>16</td>
<td>4x4</td>
<td>40.45</td>
<td></td>
</tr>
<tr>
<td>Events</td>
<td>16</td>
<td>4x4</td>
<td>26.2</td>
<td>x1.5 faster</td>
</tr>
</tbody>
</table>
Coarray-related updates - collectives

• CO_SUM, CO_MIN, CO_MAX and CO_BROADCAST originally implemented under TS18508:2015.

• F2018 spec is different to the TS. CCE collectives do not fully conform to F2018 yet - these 4 calls conform to F2018 and are supported in CCE 9:

    call co_sum( i )
    call co_min( i )
    call co_max( i )
    call co_broadcast( i, 4 )

• CO_REDUCE not implemented yet.
Coarray-related updates - atomics

• ATOMIC_DEFINE and ATOMIC_REF (introduced in F2008) have optional STAT argument in F2018.

• 9 more atomics integrated from TS18508:2015 into F2018: ATOMIC_ADD, ATOMIC_AND, ATOMIC_CAS, ATOMIC_FETCH_ADD, ATOMIC_FETCH_AND, ATOMIC_FETCH_OR, ATOMIC_FETCH_XOR, ATOMIC_OR, and ATOMIC_XOR

• Note that ATOMIC_INT_KIND and ATOMIC_LOGICAL_KIND are used to define the atomic operation targets and for example allow network hardware to be used.

• All supported in CCE9.
OTHER FEATURES
Other features – supported in CCE9 (1)

• **IMPLICIT NONE (TYPE, EXTERNAL)**
• Referencing a property of an object in a constant expression.
• d0.d, e0.d, es0.d, en0.d, g0.d and ew.de0 edit descriptors
• **STOP, ERROR STOP** now accept a scalar character expression as the stop code. The **QUIET** specifier is a scalar logical expression.
• **GET_COMMAND, GET_COMMAND_ARGUMENT** and **GET_ENVIRONMENT_VARIABLE** accept an optional argument **ERRMSG**
• **OUT_OF_RANGE( X, MOLD [,ROUND] )**
• **REDUCE( ARRAY, OPERATION [,MASK, IDENTITY, ORDERED] )** or **REDUCE( ARRAY, OPERATION, DIM[, MASK, IDENTITY, ORDERED] )** -- a new transformational intrinsic. **OPERATION** is a pure function. (*Added to provide on-image routine analogous to the coarray version.*)
Other features – supported in CCE9 (2)

- **COSHAPE** (COARRAY [,KIND] ) – for consistency with LCOBOUND / UCOBOUND.

- **RANDOM_INIT** (REPEATABLE, IMAGE DISTINCT) – major feature
  
  - CALL RANDOM_INIT(.false., .true.)
    gets same/different sequenced on each image
    and same/different sequences per program execution

- Simplification of calls of the intrinsic cmplx – these are conforming in F2018:
  **CMPLX( X [,KIND] )** or **CMPLX( X [,Y, KIND] )** – no need for KIND keyword.

- **ALL( MASK ) or ALL( MASK, DIM )** – two overloaded forms. Same for **ANY, NORM2, PARITY, and THIS IMAGE**

- Hexadecimal I/O (ex edit descriptor, gives values like -0X1.F400P+003 )
Other features – supported in CCE9 (3)

• The value `IEEE_AWAY` has been added to correspond to IEEE \( \text{roundTiesToAway} \). `IEEE_NEAREST` now corresponds to IEEE \( \text{roundTiesToEven} \).

• `IEEE_REAL( A [, KIND] )` – conversion to real using IEEE rules.

• `IEEE_RINT( A [,ROUND] )` – new optional argument `ROUND` of type `IEEE_ROUND_TYPE`.

• `IEEE_FMA( A, B, C )` – fused multiply-add added to IEEE_ARITHMETIC module.
Example: referencing a property of an object in a constant expression.

```fortran
integer :: b = bit_size(b), i
real    :: e = sqrt(sqrt(epsilon(e)))
integer :: seq(10) = [ ( i, i = 1, size(seq,1) ) ]
   write (*,*) "b: ", b
   write (*,*) "e: ", e
   write (*,*) "seq: ", seq

OUT:

b: 32
e: 1.858136058E-2
seq: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
```
Example: d0.d, e0.d, es0.d, en0.d, g0.d and ew.de0 edit descriptors

```python
print "('e=',es0.15)" , epsilon(x)
print "('e=',es0.10)" , epsilon(x)
print "('e=',es0.5)" , epsilon(x)
print "('e=',es0.2)" , epsilon(x)
```

OUT:

```
e=1.192092895507813E-07
```

```
e=1.1920928955E-07
```

```
e=1.19209E-07
```

```
e=1.19E-07
```
Example: **STOP**, **ERROR STOP** with a scalar character expression as the stop code. **QUIET** specifier is a scalar logical expression

```fortran
Integer, parameter :: msglevel = 2
character(:), allocatable :: message
message="reload"
stop message, quiet = ( msglevel < 2 )

OUT:

STOP reload
```
Example: \texttt{OUT\_OF\_RANGE( X, MOLD [,ROUND] )}

\begin{verbatim}
use, intrinsic :: iso_fortran_env
real( kind=real128 ) :: r128
integer :: ival

r128 = exp( sin( 3.555_real128 ))
write (*,*) "r128: ", r128
write (*,*) out_of_range( r128, ival, .true. )

r128 = exp ( 10000.0_real128 )
write (*,*) "r128: ", r128
write (*,*) out_of_range( r128, ival, .true. )

OUT:
  r128:  0.66916000843048095703125
  F
  r128:  8.8068182256629215872614960076445606E+4342
  T
\end{verbatim}
Example: \texttt{REDUCE( ARRAY, OPERATION [,MASK, IDENTITY, ORDERED] )}

\begin{verbatim}
real    :: arr(3,3) = 1.0e0
logical :: mask(3,3) = .true.
write (*,*) reduce( array=arr, operation=add, &
   mask=mask, identity=-3.0e0, ordered=.true. )
contains
pure real function add( x, y )
   real, intent(in) :: x, y
   add = x+y
end function add

OUT:

  9.
\end{verbatim}
Example: \texttt{COSHAPE( COARRAY [,KIND] )}

\begin{verbatim}
integer :: coar(3)[2,*]
write (*,*), "coshape( coar ):", coshape( coar )

srun -n 16 ./a.out

OUT:

coshape( coar ): 2, 8
\end{verbatim}
Example: **IEEE RINT( A [,ROUND] )**

```fortran
$ cat z.f90
use, intrinsic :: ieee_arithmetic
write (*,*) ieee_rint( -1.5, ieee_away )
write (*,*) ieee_rint( -1.5, ieee_nearest )
write (*,*) ieee_rint( -1.5, ieee_up )
write (*,*) ieee_rint( -1.5, ieee_down )
write (*,*) ieee_rint( -1.5, ieee_to_zero )
end
$ ftn z.f90
$ ./a.out
-2.
-2.
-1.
-2.
-1.
```
Other features – not yet supported in CCE9

• IEEE subnormal (denormal) features. Denormal numbers are not supported on Cray hardware. The IEEE_SUPPORT_DENORMAL inquiry function returns .false. for all kinds of arguments.

• "subnormal" instead of "denormal"

• IEEE_MODES_TYPE derived type, IEEE_GET_MODES and IEEE_SET_MODES subroutines
QUESTIONS?