Changes to the draft Fortran 2008 standard since the 2008 AGM

John Reid,

Convener ISO Fortran Working Group

Abstract

A draft of the Fortran 2008 standard was out for country comments, with a deadline of 31 August 2008. The comments were considered at the Tokyo meeting of WG5 in November 2008.

We describe the main changes that were agreed. It was decided that no further technical changes be made unless a serious flaw is found.

BCS Fortran
11 June 2009

Locks

Locks control access to data that is referenced or defined by more than one image.

A lock is a scalar variable of the derived type `lock_type` . It must be a coarray or a subobject of a coarray. It is either

1. ‘locked’ by an image, or

2. ‘unlocked’ – its initial value.

The only way to change the value is by an image executing a `lock` statement or by the image that locked it executing an `unlock` statement.

The example illustrates the use. Each image has its own queue; any image can add a task to any queue.

```
module queue_manager
  use iso_fortran_env,only:lock_type
  type task
      :
  end type
  type(lock_type) :: q_lock[*]
  type(task) :: q(100)[*]
  integer :: q_size[*]
contains
  subroutine get_task(job)
    type(task),intent(out) :: job
    lock(q_lock)
      job=q(q_size)
      q_size=q_size-1
    unlock(q_lock)
  end subroutine get_task
  subroutine put_task(job,image)
    type(task),intent(in) :: job
    integer,intent(in) :: image
    lock(q_lock[image])
      q_size[image]=q_size[image]+1
      q(q_size[image])[image] = job
    unlock(q_lock[image])
  end subroutine put_task
end module queue_manager
```

If a `lock` statement is executed for a lock variable that is locked by another image, the image waits for the lock to be unlocked by that image.

There is a form of the `lock` statement that avoids a wait when the lock variable is locked:

```
logical :: success
lock(q_lock,acquired_lock=success)
```

If the variable is unlocked, it is locked and the value of `success` is set to true; otherwise, `success` is set to false and there is no wait.

Note that a critical section

```
   critical
      :
   end critical
```

is just like a pair of `lock` and `unlock` statements with its own lock variable.

Atomic subroutines

The role of volatile coarrays in a spin-wait loop has been replaced by atomic variables whose values can be changed only by special subroutines.

```
use, intrinsic :: iso_fortran_env
logical(atomic_logical_kind) :: &
                       alk[*]=.true.
logical :: val
integer :: iam, p, q
   :
iam = this_image()
if (iam == p) then
   sync memory
   call atomic_define(alk[q],.false.)
else if (iam == q) then
   val = .true.
   do while (val)
      call atomic_ref(val,alk)
   end do
   sync memory
end if
```

Volatile variables

For coarrays, `volatile` now as for non-coarrays.

An object that is associated with a coarray has the `volatile` attribute if and only if the coarray has the `volatile` attribute.

Volatile variables are not permitted in a pure procedure.

Generalized output editing

For output of real or complex data that is not an IEEE infinity or NaN, the `G0` and `G0.`*d* edit descriptors follow the rules for the `G`*w*.*d*E*e* edit descriptor, except that any leading or trailing blanks are removed. Reasonable processor-dependent values of *w*, *d* (if not specified), and *e* are used with each output value.

Minor changes

A large number of minor changes were agreed.