Clause 4.8. Para 1. Remove angle brackets around “event post” and “event wait” and set in courier font.

Clause 4.8. Remove final sentence of para 1 and add new para:

“The lock and unlock statements provide another mechanism for ensuring that data on an image are accessed by one image at a time. A lock is a scalar variable of the derived type lock\_type that is defined in the intrinsic module iso\_fortran\_env. A lock must be a coarray or a subobject of a coarray. It has one of two states: *locked* and *unlocked*. The only way to change the value of a lock is by executing a lock or unlock statement. If a lock variable is locked, it can be unlocked only by the image that locked it. If a lock statement is executed for a lock variable that is locked by another image, the image normally waits for the lock to be unlocked by that image but there is an option to continue execution in this case. An error condition occurs for a lock statement if the lock variable is already locked by the executing image, and for an unlock statement if the lock variable is not already locked by the executing image. Here is a simple example of the use of a lock:

 lock (stack\_lock[p])

 stack\_size[p] = stack\_size[p] + 1

 stack(stack\_size[p])[p] = job

 unlock (stack\_lock[p])

Several images may execute this code at the same time but no two can be altering data on the same image p at the same time. The same data protection could be achieved with a critical construct:

 critical

 stack\_size[p] = stack\_size[p] + 1

 stack(stack\_size[p])[p] = job

 end critical

but this would prevent several images executing the code at the same time for different values of p.”

Clause 4.8. Para 4. In final sentence change “other segment” to “another segment”.

Clause 4.8. Para 5. What more do you want to know about atomics?

Clause 4.8. Add new para 6:

“Another exception for the segment ordering rule is that a variable may be declared as asynchronous. This indicates that the variable might be referenced or defined by non-Fortran procedures. It is initiated by execution of a communication initiation procedure and completed by execution of a corresponding communication completion procedure. The programmer is responsible for ensuring that the variable is not referenced between execution of an input communication initiation procedure and execution of the corresponding communication completion procedure; and not defined between execution of an output communication initiation procedure and execution of the corresponding communication completion procedure. The main reason for adding asynchronous is for interoperating with parallel-processing packages such as MPI that have procedures for nonblocking transfer of data from one process to another.”

Clause 4.8. Replace the current para 6 by:

“A further exception for the segment ordering rule is that a variable may be declared as volatile. This indicates to the compiler that, at any time, the variable might be changed and/or examined from outside the Fortran program. The feature needs to be used with care. If two processes access the variable at the same time, an inconsistent value might be obtained. The original reason for adding volatile was for interoperating with parallel-processing packages such as MPI, but it was realized that asynchronous, included for asynchronous IO, could be extended to provide a safer mechanism.”

Clause 4.8. Add new penultimate para:

"It is optional for a Fortran system to support continued execution in the presence of

failed images. If an image is regarded by the system as failed, it remains failed until execution terminates. The constant stat\_failed\_image in the intrinsic module iso\_fortran\_env is positive if failed image handling is supported and negative otherwise. If it is positive, it is used for the value of a stat= specifier or stat argument if a failed image is involved in an image control statement, a reference to an object with cosubscripts, or an invocation of a collective subroutine or atomic subroutine, and no other error condition occurs. The intrinsic function image\_status provides a test for the failure of a specified image and the intrinsic function failed\_images returns an array of image indices of failed images in the current team."

Clause 4.8. Replace the final para by:

“The do concurrent construct permits concurrent execution but does not give the user visibility or control of separate threads of execution performing the operations. By using this construct, the programmer asserts that there are no interdependencies between loop iterations. It involves the execution of a single image and the processor is responsible for organizing the use of threads or other mechanisms such as pipelining.”

Clause 6.63.1. Add at the end of the sentence: “, except when locks are used as described in Clause 4.8.”