Subject: Integration (feature creep?): fleshing out DO CONCURRENT functionality
From: Van Snyder

1 Introduction

Co-arrays are a good solution for some, but not all, classes of parallel programming problems: many problems do not have the sort of regular SPMD parallel structure to which co-arrays are most applicable. Rather, they have irregular sorts of parallelism that are more suited to the DO CONCURRENT construct, or a PARALLEL construct.

To exploit optimally a DO CONCURRENT construct with limits given by initialization expressions, in which the body is a case selector that selects according to the induction variable of the construct, a processor must do exactly the same calculations as would be necessary to exploit a PARALLEL construct optimally. A PARALLEL construct might thereby be dismissed as "mere syntax sugar," but syntax sugar reduces both development cost and ongoing maintenance cost, so it should not be dismissed.

To admit aggressive optimizations, substantial restrictions are placed upon the body of a DO CONCURRENT construct that may be undesirable to impose upon a PARALLEL construct.

The restriction that procedures executed from within a DO CONCURRENT construct shall be pure procedures could be relaxed if the CRITICAL construct were suitably extended.

2 Proposals

2.1 A PARALLEL construct

Provide a PARALLEL construct, having at least the functionality that can be gotten more verbosely and more cryptically (therefore with more fragility and more ongoing maintenance expense) by embedding a SELECT CASE construct within a DO CONCURRENT construct, e.g.,

```
PARALLEL                                DO CONCURRENT I = 1, N
                                            SELECT CASE ( I )
FORK                                        CASE ( 1 )
    block                                       block
FORK                                        CASE ( 2 )
    block                                       block
...                                         ...
                                            END SELECT
END PARALLEL                            END DO CONCURRENT
```

where each *block* in either construct can be executed concurrently with another one, or in any order with respect to another one. Indeed, a processor may ignore the parallel aspects of the PARALLEL construct. The construct itself cannot be ignored because an EXIT statement may belong to it. This, however, amounts to treating it very much like a BLOCK construct.

2.2 Exclusive access to shared variables

To provide for exclusive access to shared variables, generalize the CRITICAL construct to provide that the execution sequence of a single iteration of a DO CONCURRENT construct cannot enter it if one is already executing it. This would allow CRITICAL constructs to invoke impure procedures. This doesn't work for PARALLEL constructs, however: While different iterations of a DO CONCURRENT construct might encounter the same textual CRITICAL construct, different forks of a PARALLEL construct of necessity would not. The desired effect — exclusive access to shared variables — can be simulated by putting the CRITICAL construct into a procedure. The VALUE attribute must be implemented differently (more complicated, more expensive) from the obvious way to make it thread safe. If executable per-invocation initializations are someday provided, procedures exploiting them also would not be thread safe. The reason in both cases is that the specification part necessarily would not be within a CRITICAL construct. Therefore, it would be useful to have a MONITOR prefix for a procedure. It would furthermore be useful in connection with co-arrays. For lighter-weight synchronization, it would be useful to have a LOCK construct based upon an object of SEMAPHORE type, that type being defined in the ISO_FORTRAN_ENV intrinsic module.

2.3 Iteration-private and thread-private variables

DO CONCURRENT constructs would benefit from iteration-private variables, and blocks in a FORK construct would benefit from thread-private variables. To cater for this, allow declarations in DO CONCURRENT constructs, and in each FORK of a PARALLEL construct. This proliferation of special cases suggests it would be easier, both for processors and for the standard, simply to allow a *specification-part* in every *block*. An entity declared in the *specification-part* of a *block* would have a scope of the *block*. Allow the induction variable of a DO or DO CONCURRENT construct to be preceded by INTEGER [(*kind-selector*)] ::, having the effect of giving the induction variable construct scope, and allow it before an index variable in a FORALL construct or statement for documentary purposes, to specify the type of the index variable if it would not have integer type in the containing scope, or to specify its kind.

3 Edits

Edits refer to 06-007. Page and line numbers are displayed in the margin. Absent other instructions, a page and line number or line number range implies all of the indicated text is to be replaced by associated text, while a page and line number followed by + (-) indicates that associated text is to be inserted after (before) the indicated line. Remarks are noted in the margin, or appear between [and] in the text.

or *parallel-construct* 11:11+

15:22+

(3a) Exection of a PARALLEL construct divides the execution sequence into a number of execution sequences that does not exceed the number of FORK blocks of the construct. Each such execution sequence proceeds independently through one or more different FORK blocks of the PARALLEL construct until each FORK block of the construct has been executed exactly once, at which instant they are recombined into a single execution sequence.

(3b) Exection of a DO CONCURRENT construct divides the execution sequence into a number of execution sequences that does not exceed the iteration count of the construct. Each such execution sequence proceeds independently through the block of one or more different iterations of the construct until every iteration of the construct has been executed exactly once, at which instant they are recombined into a single execution sequence.

[Editor: Insert "END LOCK" into the table in alphabetical order.] 30:2+

[Editor: Insert "END PARALLEL" into the table in alphabetical order.]

R754 *forall-header* **is** ([INTEGER [*kind-selector*] ::] *forall-triplet-spec-list* ■ 168:33
■ [, *scalar-mask-expr*])

R801 *block* **is** [*declaration-construct*] ... 175:14
[*execution-part-construct*] ...

C800a (R801) The *declaration-construct* shall not be an *entry-stmt*.

8.1.4a PARALLEL construct 180:1-

The PARALLEL construct divides the execution sequence into a number of execution sequences that does not exceed the number of FORK blocks within the construct. Each such execution sequence independently executes one or more different *fork-block*s of the construct. These independent execution sequences recombine into a single execution sequence when each *fork-block* has been executed exactly once.

R815.1 *parallel-construct* **is** parallel-stmt
fork-block
[*fork-block*] ...

end-parallel-stmt

R815.2 *parallel-stmt* **is** [*parallel-construct-name* :] PARALLEL

R815.3 *fork-block* **is** FORK [*parallel-construct-name*]
block

C807.1 (R815.3) A *fork-block* shall not contain an EXIT or CYCLE statement that belongs to a construct that contains the parallel construct.

C807.2 (R815.3) A branch (8.2) within a *fork-block* shall not have a branch target that is outside the *parallel-construct*.

C807.3 (R815.3) A procedure referenced within a *fork-block* shall be a pure procedure or a monitor procedure, or shall be executed from within the range of a CRITICAL construct or a LOCK construct.

R815.4 *end-parallel-stmt* **is** END PARALLEL [*parallel-construct-name*]

C807.4 (R815.1) If the *parallel-stmt* of a *parallel-construct* specifies a *parallel-construct-name*, the corresponding *end-parallel-stmt* shall specify the same *parallel-construct-name*. If the *parallel-stmt* of a *parallel-construct* does not specify a *parallel-construct-name*, the corresponding *end-parallel-stmt* shall not specify a *parallel-construct-name*. If a *fork-stmt* specifies a *parallel-construct-name*, the corresponding *parallel-stmt* shall specify the same *parallel-construct-name*.

NOTE 8.9a

A processor is not required to execute the individual FORK blocks of a parallel construct concurrently. Other than verifying their syntax and constraints, a processor could simply ignore the FORK statements, with the effect that the FORK blocks are executed in the order they appear.

8.1.3a LOCK construct

A LOCK construct permits an execution sequence to enter it if its lock variable has a lock status of unlocked, and does not permit the execution sequence to enter if the lock variable has a lock status of locked. When an execution sequence enters a LOCK construct, the lock status of its lock variable becomes locked. When an execution sequence completes execution of a lock construct, the lock status of its lock variable becomes unlocked. An execution sequence that is prevented from entering is not terminated; its entry is simply delayed until the execution sequence that is executing the LOCK construct completes execution of it. If several execution sequences simultaneously attempt to enter a LOCK construct, exactly one of them enters it; which one enters it is processor dependent. If several execution sequences attempt to enter a LOCK construct while another execution sequence is executing it, which one proceeds when the execution sequence that is executing it completes executing it is processor dependent.

A LOCK construct completes execution when the END LOCK statement is executed, when control is transferred by a branch within the construct to a branch target outside of the construct, when an EXIT statement that belongs to the construct or one that contains the construct is executed, or when a CYCLE statement that belongs to a construct that contains the construct is executed.

[Alternatively, a LOCK construct shall be terminated only by execution of the END LOCK statement or an EXIT statement that belongs to the construct.]

R815.5 *lock-construct* **is** *lock-stmt*
block
end-lock-stmt

R815.6 *lock-stmt* **is** [*lock-construct-name* :] LOCK *lock-variable*

R815.7 *lock-variable* **is** *scalar-variable*

C807.4 (R815.7) The type of the *lock-variable* shall be the derived type SEMAPHORE defined in the ISO_FORTRAN_ENV intrinsic module. The lock variable shall not have the ALLOCATABLE or POINTER attribute, and shall not be a subcomponent of an object that has the ALLOCATABLE or POINTER attribute..

R815.8 *end-lock-stmt* **is** END LOCK [*lock-construct-name*]

C807.5 (R815.5) If the *lock-stmt* of a *lock-construct* specifies a *lock-construct-name*, the corresponding *end-lock-stmt* shall specify the same *lock-construct-name*. If the *lock-stmt* of a *lock-construct* does not specify a *lock-construct-name*, the corresponding *end-lock-stmt* shall not specify a *lock-construct-name*.

C809a (R816) An *associate-name* shall not be an *object-name* in a *type-declaration-stmt* in the *block*, and shall not appear in any other *declaration-construct* in the *block* except as an *object-name* in an ALLOCATABLE, ASYNCHRONOUS, POINTER, TARGET, or VOLATILE statement. 181:6+

[Editor: replace "Within . . . the attribute." by "Within the *block* of an ASSOCIATE construct or any *block* of a SELECT TYPE construct, an associating entity has the ASYNCHRONOUS or VOLATILE attribute if the selector is a variable that has the attribute or if the selector is a variable and the associating entity is specified to have the attribute by an attribute specification statement within the construct. An associating entity has the TARGET attribute if the selector is a variable and has either the TARGET or POINTER attribute or is specified to have the TARGET attribute by an attribute specification statement within the construct. An associating entity may be specified by an attribute specification statement to have the ALLOCATABLE or POINTER attribute only if the selector is a variable and has that attribute. If the *selector* is allocatable and the associating entity is not, the selector shall be allocated. If the *selector* is a pointer and the associating entity is not, the selector shall be associated with a target and the associating entity becomes associated with that target. Each associating entity has the same rank as the associated selector. If the associating entity is neither allocatable nor a pointer, or is an allocated allocatable or an associated pointer, the lower bound of each dimension is the result of the LBOUND function (13.7.97) applied to the corresponding dimension of *selector*, and the upper bound is one less than the sum of the lower bound and the extent".] 181:21-27

C813a (R816) An associate name shall not be an *object-name* in a *type-declaration-stmt* in the *block*, and shall not appear in any other *declaration-construct* in the *block* except as an *object-name* in an ALLOCATABLE, ASYNCHRONOUS, POINTER, TARGET, or VOLATILE statement. 182:15+

R831 *do-variable* **is** [INTEGER [*kind-selector*] ::] *scalar-int-variable* 185:30

When a DO CONCURRENT statement is executed, a separate instance of the *block* of the DO CONCURRENT construct is created for each iteration, and the execution sequence that executes the DO CONCURRENT statement is divided into a number of execution sequences that does not exceed the iteration count. Each instance has an independent set of local unsaved data objects. Each execution sequence independently executes one or more different instances of the block in such a way that each instance is executed once. Each instance ceases to exist when execution of its iteration of the DO CONCURRENT construct completes or execution of the program is terminated. If the program is not terminated, completion of execution of the DO CONCURRENT construct recombines the execution sequences into a single execution sequence. 187:20+ New ¶

[Make the first sentence of the paragraph, the one that begins "The processor shall ensure. . . ", a separate paragraph, and replace the three instances of "image" in it by "execution sequence". Within the remainder of the paragraph, replace "image" by "execution sequence". Within NOTE 8.23 replace the first three instances of "image" in it by "execution sequence".] 192:15-19+

---

or MONITOR 320:29+

C1246a (R1229) If MONITOR appears, neither ELEMENTAL nor RECURSIVE shall appear. 326:34+

12.8 Monitor procedures

337:13+

A **monitor procedure** is a procedure that is defined by a subprogram for which MONITOR appears in the prefix of the initial subroutine statement or function statement. It does not allow an execution sequence to enter it if one has entered it but not completed execution of it. The execution sequence that is prevented from entering is not terminated; its entry is simply delayed until the execution sequence that is executing the monitor procedure completes execution of it. If several execution sequences simultaneously attempt to enter a monitor procedure, exactly one of them enters it and the others are delayed; which one enters it is processor dependent. If several execution sequences attempt to enter a monitor procedure while another execution sequence is executing it, which one proceeds when the execution sequence that is executing it completes executing it is processor dependent.

[Editor: Replace "derived type" by "derived-type definitions".] 437:30

13.8.3.5a The SEMAPHORE derived type 439:1+

The type of a *lock-variable* in a LOCK construct (8.1.3a) shall be the SEMAPHORE derived type. The SEMAPHORE derived type has private components, at least one of which has default initialization that indicates that the initial lock status of objects of SEMAPHORE derived type is unlocked.

16.4 Statement, construct and block entities 491:24

[Editor: Replace "or" by comma. After "ASSOCIATE construct" insert ", or a *do-variable* that follows INTEGER[*kind-selector*] :: in a DO construct".] 491:28-29

An entity that is declared or defined by a *declaration-construct* in a *block* is a block entity that has a scope of that *block*. 491:41+ New ¶

If a global or local identifier accessible within the scope of a block is the same as the identifier of a block entity of the block, the identifier is interpreted within the block as that of the block entity. Elsewhere in the scoping unit the identifier is interpreted as the global or local identifier. 492:31+