This allows synchronization between iterations from low to high iteration numbers.

(But not necessarily completed).

\[ i > i \text{ have been assigned to a processor} \]

\[ i \text{ is not assigned to any processor until all iterations.} \]

Restriction is placed on how the system may schedule the parallelism is allowed across the iterations. However, a doall in the sense that

The doall is the same as a doall in the sense that there is another common parallel loop construct.
end doall

call advance(1) stocal, 

call wait(1) stocal, 

stocal = sum(x(i)+31)

j = (i-1)*32 + 1

loop

double precision stocal

integer j

doall i = 1, n/32

0 = s

1x

0 =
The list of variables that must be flushed from
register to memory to be read by another processor.

- Increment by 1 (or some specified amount).
- It may, in fact, set the counter to the value of ! or
incremented with the event number given as the argument.

In advance/await pairs, on an advance, the processor
supports. Each iteration therefore can have k
number. There are typically a fixed and small number, k,
The parameter to the advance/await routines is an event
section code after the wait.

This requires for efficiency that the section before the
constraint on the iterations that enter it.

This is like a critical section but with an ordering
executed in iteration i-1.

Iteration i will not proceed until the advance has been
(or perhaps, i.e., the processor executing the wait in
given as the argument reaches the iteration value i - 1,
the system variable associated with the event number
On an await, a processor executing iteration i waits until

value in the argument.

not wait, otherwise it waits until the event exceeds the

current iteration is one of the initial iterations that do

WAIT finds a negative number it assumes that the

places the value of I in the event variable and that it an

Assume that an ADVANCE executed on iteration I

always straightforward to use.

This must be used with great care since it is not

iteration I, could wait for the counter to hit 3.

For example, the await executed on event variable I on

the processor is waiting.

argument that gives the value of the counter for which

In some cases await also has an optional second

Consider the sequential code:

\[
\begin{align*}
\text{DO} & \text{ I = 1, 100} \\
A(I) & = B(I) \times C(I) + D(I) \\
B(I) & = C(I) / D(I-1) + A(I-3) \\
\text{END DO}
\end{align*}
\]

The loop cannot be transformed to a DOALL loop.
THIS CODE DOES NOT WORK PROPERLY

end doacross

\( b(i) = c(i) \) 
\( d(i-1) + a(i-3) \)

\( \text{AMAIT}(1, i-3) \)

\( \text{ADVANCE}(1) \)

\( a(i) = b(i) + c(i) \)

\( d(i) \)

\( \text{doacross} \ i = 1 \to 100 \)

\( \text{synchronization point} \) would be as follows:

The first attempt to use the modified await and I
has not executed its advance.

iteration 4 passes the wait even though iteration 1
either to 1 or to 2 depending on implementation (and
and)

However, the advance in iteration 3 bumps the counter

the advance to be hit by iteration 1.

iteration 4 is waiting for the counter to exceed 1, i.e., for

assigned to a processor and at the wait(1) instruction.

Suppose 3 gets their first and iteration 4 is already

(initialization does not matter)

reach their advance(1) instructions. (note the

cannot make any assumptions about what order they will

when iterations 1, 2, and 3 are assigned to processors we

 Beware of the order of the advance and wait calls.

•
THIS CODE DOES NOT WORK PROPERLY

\[
\text{end doacross}
\]

\[
\begin{align*}
\text{b}(i) &= \text{c}(i) / (\text{d}(i-1) + \text{a}(i-3)) \\
\text{advance}(i) \\
\text{wait}(i', i-3) \\
\text{a}(i) &= \text{b}(i) \times \text{c}(i) + \text{d}(i) \\
\text{doacross } i &= 1..100
\end{align*}
\]

Suppose we reverse the order
Way.

Iterations are as before and failure can occur in the same way.

After that however the order of the advances for the three iterations matters, i.e., initialization matters.

The awaits will succeed for iterations 1, 2, and 3 due to
AVAIT waits for I-1 to appear in event counter

ADVANCE set event counter to I

Ignore the I-3 distance in the dependence
end doacross

\[
B(I) = C(I) \left( \frac{D(I-I) + A(I-3)}{A(I-I) + D(I)} \right)
\]

ADVANCE(I)

WAIT(I)

\[
A(I) = B(I) \times C(I) + D(I)
\]

doacross I = 1, 100
distance dependence exists
carefully enough at what happens when a not trivial

The problem is that in all cases we have not looked

The code has become essentially sequential.

Therefore there are unnecessary dependences.

But note that iteration 4 waits for 1, 2, and 3 and

This works since all dependences are enforced

•
In fact, 3 different advance/await events are required.

- (3, 6, 9, 12, 15, ...)
- (2, 5, 8, 11, 14, ...)
- (1, 4, 7, 10, 13, ...)

These are the 100 iterations split into three sets that depend on their internal iterations with no dependences between the sets.
ADVANCE(2, I)

ADVANCE(1, I)

ADVANCE(1, I-3)

WAIT(2, I-3)

ELSE IF (I MOD 3 = 2) THEN

ADVANCE(2, I)

ADVANCE(1, I)

WAIT(1, I-3)

IF (I MOD 3 = 1) THEN

A(I) = B(I) * C(I) + D(I)

doacross I = 1, 100

The following code would work:
end doacross

b(i) = c(i) / (d(i-1) + a(i-3))

end if
respecitively indicatng incremented by \( \text{ADVANCE}(3.3) \) and \( \text{ADVANCE}(3.3) \)

the form

set to -2, -1, 0 respectively and the advances could be of

Alternatively, one could assume that counters 1, 2, 3 were

been added to emphasize that it sets the counter to 1.

Note an explicit second parameter on the advance has