9.18 Run-time decisions

Sometimes all what is needed for translation into DOALL is a critical section.

The following loop has a cyclic dependence graph (output dependences)

\[
\text{do } i=1,n \\
\quad a(k(i)) = a(k(i)) + 1 \\
\text{end do}
\]

This loop can be transformed into DOALL by just inserting a critical section as shown next:

\[
\text{do } i=1,n \\
\quad \text{critical } a(k(i)) \text{ do} \\
\quad \quad a(k(i)) = a(k(i)) + 1 \\
\quad \text{end critical} \\
\text{end do}
\]
9.18.1 Handling Output Dependences at Run-Time

do i=1,n
    a(k(i)) = c(i) + 1
end do

To parallelize the following loop we create a structure for each a(i) with two components \%sync and \%data and translate into:

a(k(:))\%sync = 0

doall i=1,n
    critical a(k(i))
    if a(b(i))\%sync < i then
        a(k(i))\%data = c(i) + 1
        a(k(i))\%sync = i
    end if
end critical
end doall
Assume $k(i)$ has the following values

$$i = \begin{array}{cccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
k(i) = & 3 & 5 & 7 & 3 & 4 & 5 & 6 & 9 & 10 & 3
\end{array}$$

The critical section will be reached by all iterations. Let us assume the following order of arrival among the conflicting ones:

for $a(3)$: $4 \ 1 \ 10$

for $a(5)$: $6 \ 2$

In the previous loop $a(k(i)) \% \text{data}$ will be assigned only once for $i=3, 5, 7, 8, 9, 10$

$a(5)$ will be assigned once since when iteration 2 enters the critical section after iteration 6 leaves, $a(5) \% \text{sync}$ will be 6, and the boolean function inside the if will be false.

$a(3)$ will be assigned twice. Once for iteration 4 and once for iteration 10. No assignment takes place when iteration 1 enters the critical section after iteration 4 leaves.
9.18.2 Handling flow dependences at Run-Time

do i=1,n
    a(k(i)) = ...
    ... = a(j(i))
end do
repeat until all(done)
  doall i=1,n
    if (.not.done(i)) then
      a(k(i))%sync = ∞
      a(j(i))%sync = ∞
    end if
  end doall
  doall i=1,n
    if (.not.done(i)) then
      critical a(k(i))
      if a(k(i))%sync > i then a(k(i))%sync=i
      if a(j(i))%sync > i then a(j(i))%sync=i
    end critical
  end doall
  doall i=1,n
    if (.not.done(i)) then
      if (a(k(i))%sync = i & a(j(i))%sync = i) then
        a(k(i)) = ...
        ... = a(j(i))
        done(i) = .true.
      end if
    end if
  end doall
end repeat
On each iteration of the `repeat`, the second `doall` selects those iterations not processed, and for a collection of iterations $i_1, i_2, ... i_K$ with $k(i_1)=k(i_2)= ...=k(i_K)=K$, $a(K)\%\text{sync}$ gets the value $\min(i_1, i_2, ... i_K)$.

The third `doall` computes only those pairs where both $a(k(i))\%\text{sync}$ and $a(j(i))\%\text{sync}$ have the same value. The reason these iterations can be executed is that either no previous iterations of the original do loop reference the same array elements or earlier iterations referring to the same elements of $a$ have already been executed in previous iterations of the `repeat`. 