Data Dependence Relations

Flow or true dependence

S1: \( X = \ldots \)
S2: \( \ldots = X \)

\( \delta \)

Anti-dependence

S1: \( \ldots = X \)
S2: \( X = \ldots \)

\( \bar{\delta} \)

Output dependence

S1: \( X = \ldots \)
S2: \( X = \ldots \)

\( \delta^o \)
Data Dependence Direction Vector

- Data dependence relations are augmented with a direction of data dependence (direction vector).
- There is one direction vector component for each loop in a nest of loops.
- The *data dependence direction vector* (or direction vector) is \( \psi = (\psi_1, \psi_2, \ldots, \psi_d) \), where \( \psi_k \in \{<, =, >, \leq, \geq, \neq, *\} \).
- Forward or “<” direction means dependence from iteration \( i \) to \( i + k \) (i.e., computed in iteration \( i \) and used in iteration \( i + k \)).
- Backward or “>” direction means dependence from iteration \( i \) to \( i - k \) (i.e., computed in iteration \( i \) and used in iteration \( i - k \)). This is not possible in single loops and possible in two or higher levels of nesting.
- Equal or “=” direction means that dependence is in the same iteration (i.e., computed in iteration \( i \) and used in iteration \( i \)).
Direction Vector Example 1

for J = 1 to 100 do {
  S: X(J) = X(J) + c
}

for J = 1 to 99 do {
  S: X(J+1) = X(J) + c
}

for J = 1 to 99 do {
  S: X(J) = X(J+1) + c
}

for J = 99 downto 1 do {
  S: X(J) = X(J+1) + c
}

for J = 2 to 101 do {
  S: X(J) = X(J-1) + c
}

S $\bar{\delta}_S$ X(1) = X(1) + c
X(2) = X(2) + c

S $\delta_S$ X(2) = X(1) + c
X(3) = X(2) + c

S $\delta_S$ X(1) = X(2) + c
X(2) = X(3) + c

S $\delta_S$ X(99) = X(100) + c
X(98) = X(99) + c
note ‘-ve’ increment

S $\delta_S$ X(2) = X(1) + c
X(3) = X(2) + c

Y.N. Srikant  Automatic Parallelization
for $i = 1$ to $5$ do {
    for $j = 1$ to $4$ do {
        $S1$: $A(i, j) = B(i, j) + C(i, j)$
        $S2$: $B(i, j+1) = A(i, j) + B(i, j)$
    }
}

Demonstration of direction vector

$I = 1, J = 1$: $A(1,1) = B(1,1) + C(1,1)$
$B(1,2) = A(1,1) + B(1,1)$

$J = 2$: $A(1,2) = B(1,2) + C(1,2)$
$B(1,3) = A(1,2) + B(1,2)$

$J = 3$: $A(1,3) = B(1,3) + C(1,3)$
$B(1,4) = A(1,3) + B(1,3)$

$S1 \delta_{(=,=)} S2$
$S2 \delta_{(=,<)} S1$
$S2 \delta_{(=,<)} S2$
Direction Vector Example 3

\[ S_1 \delta_{(<,>)} S_2 \]

```
for l = 1 to N do {
    for J = 1 to N do {
        S1: A(l+1, J) = ...
        S2: ... = A(l, J+1)
    }
}
```

```
I = 1, J = 2
S1: A(2,2) = ...
I = 2, J = 1
S2: ... = A(2,2)
```

```
S2 \delta_{(<,>)} S1
```

```
for l = 1 to N do {
    for J = 1 to N do {
        S1: ... = A(l, J+1)
        S2: A(l+1, J) = ...
    }
}
```

```
I = 1, J = 2
S2: A(2,2) = ...
I = 2, J = 1
S1: ... = A(2,2)
```
Direction Vector Example 4

for $l = 1$ to 100 do {
    for $j = 1$ to 100 do {
        for $k = 1$ to 100 do {
            $S1: \quad X(l, j+1, k) = A(l, j, k) + 10$
        }
        for $l = 1$ to 50 do {
            $S2: \quad A(l+1, j, l) = X(l, j, l) + 5$
        }
    }
}

<table>
<thead>
<tr>
<th>J = 1</th>
<th>$l = 1$</th>
<th>$l = 2$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$X(1,2,K) = A(1,1,K)$</td>
<td>$X(2,2,K) = A(2,1,K)$</td>
</tr>
<tr>
<td></td>
<td>$A(2,1,L) = X(1,1,L)$</td>
<td>$A(3,1,L) = X(2,1,L)$</td>
</tr>
<tr>
<td>$\delta_{=,&lt;}$</td>
<td>$\delta_{=,&lt;}$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>J = 2</th>
<th>$l = 1$</th>
<th>$l = 2$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$X(1,3,K) = A(1,2,K)$</td>
<td>$X(2,3,K) = A(2,2,K)$</td>
</tr>
<tr>
<td></td>
<td>$A(2,2,L) = X(1,2,L)$</td>
<td>$A(3,2,L) = X(2,2,L)$</td>
</tr>
<tr>
<td>$\delta_{=,&lt;}$</td>
<td>$\delta_{=,&lt;}$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>J = 3</th>
<th>$l = 1$</th>
<th>$l = 2$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$X(1,4,K) = A(1,3,K)$</td>
<td>$X(2,4,K) = A(2,3,K)$</td>
</tr>
<tr>
<td></td>
<td>$A(2,3,L) = X(1,3,L)$</td>
<td>$A(3,3,L) = X(2,3,L)$</td>
</tr>
</tbody>
</table>
Individual nodes are statements of the program and edges depict data dependence among the statements.

If the DDG is acyclic, then vectorization of the program is possible and is straightforward.

- Vector code generation can be done using a topological sort order on the DDG.

Otherwise, find all the strongly connected components of the DDG, and reduce the DDG to an acyclic graph by treating each SCC as a single node.

- SCCs cannot be fully vectorized; the final code will contain some sequential loops and possibly some vector code.
If all the dependence relations in a loop nest have a direction vector value of “=” for a loop, then the iterations of that loop can be executed in parallel with no synchronization between iterations.

Any dependence with a forward (>) direction in an outer loop will be satisfied by the serial execution of the outer loop.

If an outer loop L is run in sequential mode, then all the dependences with a forward (>) direction at the outer level (of L) will be automatically satisfied (even those of the loops inner to L).

However, this is not true for those dependences with (=) direction at the outer level; the dependences of the inner loops will have to be satisfied by appropriate statement ordering and loop execution order.
Vectorization Example 1

for $l = 1$ to $99$ {
    \begin{align*}
    S1: & \quad X(l) = l \\
    S2: & \quad B(l) = 100 - l \\
    \end{align*}
}

for $l = 1$ to $99$ {
    \begin{align*}
    S3: & \quad A(l) = F(X(l)) \\
    S4: & \quad X(l+1) = G(B(l)) \\
    \end{align*}
}

$X(1:99) = (/1:99/)$
$B(1:99) = (/99:1:-1/)$
$X(2:100) = G(B(1:99))$
$A(1:99) = F(X(1:99))$

Loop A is parallelizable, but loop B is not, due to forward dependence of S3 on S4.
Vectorization Example 2.1

for \( i = 1 \) to 100 do 
    for \( j = 1 \) to 100 do 
        for \( k = 1 \) to 100 do 
            \( S1: \quad X(i, j+1, k) = A(i, j, k) + 10 \)
        
    for \( l = 1 \) to 50 do 
        \( S2: \quad A(i+1, j, l) = X(i, j, l) + 5 \)

<table>
<thead>
<tr>
<th>( i ) = 1</th>
<th>( i ) = 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>( j = 1 )</td>
<td>( X(1,2,K) = A(1,1,K) )</td>
</tr>
<tr>
<td></td>
<td>( A(2,1,L) = X(1,1,L) )</td>
</tr>
<tr>
<td>( j = 2 )</td>
<td>( X(1,3,K) = A(1,2,K) )</td>
</tr>
<tr>
<td></td>
<td>( A(2,2,L) = X(1,2,L) )</td>
</tr>
<tr>
<td>( j = 3 )</td>
<td>( X(1,4,K) = A(1,3,K) )</td>
</tr>
<tr>
<td></td>
<td>( A(2,3,L) = X(1,3,L) )</td>
</tr>
</tbody>
</table>
Vectorization Example 2.2

I loop cannot be vectorized due to the cycle.

I and J loops cannot be parallelized, due to '<' direction vector. K and L loops can be parallelized.

for $l = 1$ to 100 do {
    for $j = 1$ to 100 do {
        for $k = 1$ to 100 do {
            S1: $X(l, j+1, k) = A(l, j, k) + 10$
        }
        for $l = 1$ to 50 do {
            S2: $A(l+1, j, l) = X(l, j, l) + 5$
        }
    }
}

for $l = 1$ to 100 do {
    $X(l, 2:101, 1:100) = A(l, 1:100, 1:100) + 10$
    $A(l+1, 1:100, 1:50) = X(l, 1:100, 1:50) + 5$
}
Vectorization Example 2.3

If the I loop is run sequentially, the I-loop dependences are satisfied; J-loop dependences change as shown and there are no more cycles. The loops can be vectorized. However, J-loop cannot be (still) parallelized.

```
for I = 1 to 100 do {
    for J = 1 to 100 do {
        for K = 1 to 100 do {
            S1: X(I, J+1, K) = A(I, J, K) + 10
        }
        for L = 1 to 50 do {
            S2: A(I+1, J, L) = X(I, J, L) + 5
        }
    }
}
```

```
for I = 1 to 100 do {
    X(I, 2:101, 1:100) = A(I, 1:100, 1:100) + 10
    A(I+1, 1:100, 1:50) = X(I, 1:100, 1:50) + 5
}
```
Vectorization Example 2.4

For $l = 1$ to 100 do {
    for $j = 1$ to 100 do {
        for $k = 1$ to 100 do {
            $s1$: $x(l, j+1, k) = a(l, j, k) + 10$
        }
        for $l = 1$ to 50 do {
            $s2$: $a(l+1, j+1, l) = x(l, j, l) + 5$
        }
    }
}

<table>
<thead>
<tr>
<th></th>
<th>$l = 1$</th>
<th>$l = 2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$j = 1$</td>
<td>$x(1,2,k) = a(1,1,k)$</td>
<td>$x(2,2,k) = a(2,1,k)$</td>
</tr>
<tr>
<td></td>
<td>$a(2,2,l) = x(1,1,l)$</td>
<td>$a(3,2,l) = x(2,1,l)$</td>
</tr>
<tr>
<td>$j = 2$</td>
<td>$x(1,3,k) = a(1,2,k)$</td>
<td>$x(2,2,k) = a(2,2,k)$</td>
</tr>
<tr>
<td></td>
<td>$a(2,3,l) = x(1,2,l)$</td>
<td>$a(3,3,l) = x(2,2,l)$</td>
</tr>
<tr>
<td>$j = 3$</td>
<td>$x(1,4,k) = a(1,3,k)$</td>
<td>$x(2,4,k) = a(2,3,k)$</td>
</tr>
<tr>
<td></td>
<td>$a(2,4,l) = x(1,3,l)$</td>
<td>$a(3,4,l) = x(2,3,l)$</td>
</tr>
</tbody>
</table>
Vectorization Example 2.5

If the program is changed slightly, then dependences change as shown. I and J loops are not parallelizable. If I and J loops are interchanged and J-loop is run sequentially, I-loop can be parallelized. K and L loops are always parallelizable.

```
for l = 1 to 100 do {
    for J = 1 to 100 do {
        for K = 1 to 100 do {
            S1:    X(l, J+1, K) = A(l, J, K) + 10
                    }
            for L = 1 to 50 do {
                S2:    A(l+1, J+1, L) = X(l, J, L) + 5
                        }
        }
    }
}
```

```
for l = 1 to 100 do {
    X(l, 2:101, 1:100) = A(l, 1:100, 1:100) + 10
    A(l+1, 2:101, 1:50) = X(l, 1:100, 1:50) + 5
}
```
Vectorization Example 2.6

Before interchange

\[ \delta_{<,<} \]

\[ \delta_{=,<} \]

\[ S1 \]

\[ S2 \]

After interchange

\[ \delta_{<,<} \]

\[ \delta_{<,=} \]

\[ S1 \]

\[ S2 \]

for \( J = 1 \) to 100 do {
  for \( L = 1 \) to 100 do {
    for \( K = 1 \) to 100 do {
      \( S1: \quad X(I, J+1, K) = A(I, J, K) + 10 \)
    }
    for \( L = 1 \) to 50 do {
      \( S2: \quad A(I+1, J+1, L) = X(I, J, L) + 5 \)
    }
  }
}

<table>
<thead>
<tr>
<th></th>
<th>( I = 1 )</th>
<th>( I = 2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( J = 1 )</td>
<td>( X(1,2,K) = A(1,1,K) )</td>
<td>( X(2,2,K) = A(2,1,K) )</td>
</tr>
<tr>
<td></td>
<td>( A(2,2,L) = X(1,1,L) )</td>
<td>( A(3,2,L) = X(2,1,L) )</td>
</tr>
<tr>
<td>( J = 2 )</td>
<td>( X(1,3,K) = A(1,2,K) )</td>
<td>( X(2,2,K) = A(2,2,K) )</td>
</tr>
<tr>
<td></td>
<td>( A(2,3,L) = X(1,2,L) )</td>
<td>( A(3,3,L) = X(2,2,L) )</td>
</tr>
<tr>
<td>( J = 3 )</td>
<td>( X(1,4,K) = A(1,3,K) )</td>
<td>( X(2,4,K) = A(2,3,K) )</td>
</tr>
<tr>
<td></td>
<td>( A(2,4,L) = X(1,3,L) )</td>
<td>( A(3,4,L) = X(2,3,L) )</td>
</tr>
</tbody>
</table>
Concurrentization Examples

for I = 2 to N do {
    for J = 2 to N do {
        S1: A(I,J) = B(I,J) + 2
        S2: B(I,J) = A(I-1, J-1) - B(I,J)
    }
}

S1 \delta_{(<,)} S2, S1 \delta_{(=,)} S2, S2 \delta_{(=,)} S2

<table>
<thead>
<tr>
<th>I = 1</th>
<th>I = 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>J = 1</td>
<td></td>
</tr>
<tr>
<td>A(2,2)=</td>
<td>A(3,2)=</td>
</tr>
<tr>
<td>= A(1,1)</td>
<td>= A(2,1)</td>
</tr>
<tr>
<td>J = 2</td>
<td></td>
</tr>
<tr>
<td>A(2,3)=</td>
<td>A(3,3)=</td>
</tr>
<tr>
<td>= A(1,2)</td>
<td>= A(2,2)</td>
</tr>
<tr>
<td>J = 3</td>
<td></td>
</tr>
<tr>
<td>A(2,4)=</td>
<td>A(3,4)=</td>
</tr>
<tr>
<td>= A(1,3)</td>
<td>= A(2,3)</td>
</tr>
</tbody>
</table>

If the I loop is run in serial mode then, the J loop can be run in parallel mode.

for I = 2 to N do {
    for J = 2 to N do {
        S1: A(I,J) = B(I,J) + 2
        S2: B(I,J) = A(I, J-1) - B(I,J)
    }
}

S1 \delta_{(=,)} S2, S1 \delta_{(=,)} S2, S2 \delta_{(=,)} S2

<table>
<thead>
<tr>
<th>I = 1</th>
<th>I = 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>J = 1</td>
<td></td>
</tr>
<tr>
<td>A(2,2)=</td>
<td>A(3,2)=</td>
</tr>
<tr>
<td>= A(2,1)</td>
<td>= A(3,1)</td>
</tr>
<tr>
<td>J = 2</td>
<td></td>
</tr>
<tr>
<td>A(2,3)=</td>
<td>A(3,3)=</td>
</tr>
<tr>
<td>= A(2,2)</td>
<td>= A(3,2)</td>
</tr>
<tr>
<td>J = 3</td>
<td></td>
</tr>
<tr>
<td>A(2,4)=</td>
<td>A(3,4)=</td>
</tr>
<tr>
<td>= A(2,3)</td>
<td>= A(3,3)</td>
</tr>
</tbody>
</table>

The J loop cannot be run in parallel mode. However, the I loop can be run in parallel mode.
Loop Transformations for increasing Parallelism

- Recurrence breaking
  - Ignorable cycles
  - Scalar expansion
  - Scalar renaming
  - Node splitting
  - Threshold detection and index set splitting
  - If-conversion

- Loop interchanging

- Loop fission

- Loop fusion
Scalar Expansion

Not vectorizable or parallelizable

\[
\text{for } l = 1 \text{ to } N \text{ do } \{
\begin{align*}
\text{S1: } & T = A(l) \\
\text{S2: } & A(l) = B(l) \\
\text{S3: } & B(l) = T
\end{align*}
\}
\]

Vectorizable due to scalar expansion

\[
\text{for } l = 1 \text{ to } N \text{ do } \{
\begin{align*}
\text{S1: } & Tx(l) = A(l) \\
\text{S2: } & A(l) = B(l) \\
\text{S3: } & B(l) = Tx(l)
\end{align*}
\}
\]

Parallelizable due to privatization

\[
\text{forall } l = 1 \text{ to } N \text{ do } \{
\begin{align*}
\text{private temp } & S1: \text{ temp = A(l)} \\
\text{S1: } & \text{ temp = A(l)} \\
\text{S2: } & A(l) = B(l) \\
\text{S3: } & B(l) = \text{ temp}
\end{align*}
\}
\]

Acyclic DDG
Scalar Expansion is not always profitable

Not vectorizable or parallelizable

for $l = 1$ to $N$ do 
  S1: $T = T + A(l) + A(l+2)$
  S2: $A(l) = T$

Not vectorizable even after scalar expansion

for $l = 1$ to $N$ do 
  S1: $Tx(l) = Tx(l-1) + A(l) + A(l+2)$
  S2: $A(l) = Tx(l)$

Cyclic DDG

Still cyclic DDG
Scalar Renaming

The output dependence S1 5^o S3 cannot be broken by scalar expansion

for l = 1 to N do {
    S1: T = A(l) + B(l)
    S2: C(l) = T*2
    S3: T = D(l) * B(l)
    S4: A(l+2) = T + 5
}

The output dependence S1 5^o S3 CAN be broken by scalar renaming

for l = 1 to N do {
    S1: T1 = A(l) + B(l)
    S2: C(l) = T1*2
    S3: T2 = D(l) * B(l)
    S4: A(l+2) = T2 + 5
}

S3: T2(1:100) = D(1:100) * B(1:100)
S4: A(3:102) = T2(1:100) + 5(1:100)
S1: T1(1:100) = A(1:100) + B(1:100)
S2: C(1:100) = T1(1:100)*2(1:100)
T = T2(100)

5(1:100) and 2(1:100) are vectors of constants
If-Conversion

\[
\begin{align*}
\text{for } l = 1 \text{ to } 100 \text{ do } \{ & \\
& \text{if } (A(l) \leq 0) \text{ then continue} \\
& \text{end if} \\
& A(l) = B(l) + 3 \\
\} \\
\text{for } l = 1 \text{ to } 100 \text{ do } \{ & \\
& B(R(l)) = (A(l) \leq 0) \\
& \text{if } \sim B(R(l)) \text{ then} \\
& \text{end if} \\
& A(l) = B(l) + 3 \\
\} \\
B(R(1:N)) = (A(1:N) \leq 0) \\
\text{where } (\sim B(R(1:N))) \\
A(1:N) = B(1:N) + 3
\end{align*}
\]

\[
\begin{align*}
\text{for } l = 1 \text{ to } N \text{ do } \{ & \\
& S1: A(l) = D(l) + 1 \\
& S2: \text{if } (B(l) > 0) \text{ then} \\
& S3: C(l) = C(l) + A(l) \\
& S4: D(l+1) = D(l+1) + 1 \\
& \text{end if} \\
\} \\
\text{for } l = 1 \text{ to } N \text{ do } \{ & \\
& S2: \text{temp}(1:N) = B(1:N) > 0 \\
& S4: \text{where } (\text{temp}(1:N)) \\
& \quad D(2:N+1) = D(2:N+1) + 1 \\
& S1: A(1:N) = D(1:N) + 1 \\
& S3: \text{where } (\text{temp}(1:N)) \\
& \quad C(1:N) = C(1:N) + A(1:N) \\
\} \\
\end{align*}
\]
For machines with vector instructions, inner loops are preferrable for vectorization, and loops can be interchanged to enable this.

For multi-core and multi-processor machines, parallel outer loops are preferred and loop interchange may help to make this happen.

Requirements for simple loop interchange:

1. The loops L1 and L2 must be tightly nested (no statements between loops).
2. The loop limits of L2 must be invariant in L1.
3. There are no statements $S_v$ and $S_w$ (not necessarily distinct) in L1 with a dependence $S_v \delta^{(<=,>)} S_w$. 

Y.N. Srikant

Automatic Parallelization
Loop Interchange for Vectorizability

for \( l = 1 \) to \( N \) do {
    for \( j = 1 \) to \( N \) do {
        S: \( A(l, j+1) = A(l, j) \ast B(l, j) + C(l, j) \)
    }
}

Inner loop is not vectorizable
\[ S \delta_{(=,<)} S \]

for \( j = 1 \) to \( N \) do {
    for \( l = 1 \) to \( N \) do {
        S: \( A(l, j+1) = A(l, j) \ast B(l, j) + C(l, j) \)
    }
}

Inner loop is vectorizable
\[ S \delta_{(<,=)} S \]

for \( j = 1 \) to \( N \) do {
    S: \( A(1:N, j+1) = A(1:N, j) \ast B(1:N, j) + C(1:N, j) \)
}
Loop Interchange for parallelizability

for \( l = 1 \) to \( N \) do {
    for \( j = 1 \) to \( N \) do {
        \( S: \quad A(l+1,j) = A(l,j) \times B(l,j) + C(l,j) \)
    }
}

Outer loop is not parallelizable, but inner loop is

\( S \delta_{(<,=)} S \)
Less work per thread

for \( j = 1 \) to \( N \) do {
    for \( l = 1 \) to \( N \) do {
        \( S: \quad A(l+1,j) = A(l,j) \times B(l,j) + C(l,j) \)
    }
}

Outer loop is parallelizable but inner loop is not

\( S \delta_{(=,<)} S \)
More work per thread

forall \( j = 1 \) to \( N \) do {
    for \( l = 1 \) to \( N \) do {
        \( S: \quad A(l+1,j) = A(l,j) \times B(l,j) + C(l,j) \)
    }
}

Y.N. Srikant
Automatic Parallelization
Legal Loop Interchange

S11 → S12 → S13
S21 → S22 → S23
S31 → S32 → S33

dependence
loop exec order
before interchange
loop exec order
after interchange

S δ(=,<) S
Illegal Loop Interchange

\[ S \delta_{(\langle,\rangle)} S \]

dependence

loop exec order before interchange

loop exec order after interchange

Srikant

Automatic Parallelization
Legal but not beneficial Loop Interchange

\[ S \delta_{(=,<)} S \quad \text{and} \quad S \delta_{(<,=)} S \]

dependence  loop exec order  loop exec order
before interchange   after interchange
Loop Fission - Motivation

for \( l = 1 \) to \( N \) do 
\begin{align*}
S1: & \quad A(l) = E(l) + 1 \\
S2: & \quad B(l) = F(l) \times 2 \\
S3: & \quad C(l+1) = C(l) \times A(l) + D(l) \\
S4: & \quad D(l+1) = C(l+1) \times B(l) + D(l)
\end{align*}

}\}

The above loop cannot be vectorized

for \( l = 1 \) to \( N \) do 
\begin{align*}
L1: & \quad A(l) = E(l) + 1 \\
S2: & \quad B(l) = F(l) \times 2 \\
\end{align*}

}\}

for \( l = 1 \) to \( N \) do 
\begin{align*}
L2: & \quad C(l+1) = C(l) \times A(l) + D(l) \\
S4: & \quad D(l+1) = C(l+1) \times B(l) + D(l)
\end{align*}

}\}

L1 can be vectorized, but L2 cannot be
Loop Fission: Legal and Illegal

\[
\text{for } i = 1 \text{ to } N \text{ do }
\{
\begin{align*}
S1: & \quad A(i) = D(i) \times T \\
S2: & \quad B(i) = (C(i) + E(i))/2 \\
S3: & \quad C(i+1) = A(i) + 1
\end{align*}
\}
\]

In the above loop, \(S3 \prec \delta \prec S2\), and \(S3\) follows \(S2\). Therefore, cutting the loop between \(S2\) and \(S3\) is illegal. However, cutting the loop between \(S1\) and \(S2\) is legal.

\[
\text{for } i = 1 \text{ to } N \text{ do }
\{
\begin{align*}
S1: & \quad A(i+1) = B(i) + D(i) \\
S2: & \quad B(i) = (A(i) + B(i))/2 \\
S3: & \quad C(i) = B(i) + 1
\end{align*}
\}
\]

The above loop can be cut between \(S1\) and \(S2\), and also between \(S2\) and \(S3\).