Data dependences

Fundamental execution ordering constraints:

S1: \( a = b + c \)
S2: \( d = a \times 2 \)
S3: \( a = c + 2 \)
S4: \( e = d + c + 2 \)

- S1 must execute before S2 (flow-dependence)
- S2 must execute before S3 (anti-dependence)
- S1 must execute before S3 (output-dependence)
- But S3 and S4 can execute concurrently

Types of dependences

Three types are usually defined:

flow-dependence occurs when a variable which is assigned a value in one statement say \( S_1 \) is used in another statement, say \( S_2 \) later. Written as \( S_1 \delta f S_2 \).

anti-dependence occurs when a variable which is used in one statement say \( S_1 \) is assigned a value in another statement, say \( S_2 \), later. Written as \( S_1 \delta a S_2 \).

output-dependence occurs when a variable which is assigned a value in one statement say \( S_1 \) is later reassigned in another statement, say \( S_2 \). Written as \( S_1 \delta o S_2 \).
Types of dependences - continued

Type of dependence found by using IN and OUT sets for each statement:

IN(S): the set of memory locations read by the statement S

OUT(S): the set of memory locations written by the statement S. A memory location may ∈ both IN(S) and OUT(S)

If $S_1$ is executed before $S_2$ in sequential execution, then

- $\text{OUT}(S_1) \cap \text{IN}(S_2) \neq \{\} \Rightarrow S_1 \delta f S_2$
- $\text{IN}(S_1) \cap \text{OUT}(S_2) \neq \{\} \Rightarrow S_1 \delta a S_2$
- $\text{OUT}(S_1) \cap \text{OUT}(S_2) \neq \{\} \Rightarrow S_1 \delta o S_2$

Data dependence in loops

Associate an instance to each statement. For example

for $i = 1$ to $50$ do

$S_1$: $A(i) = B(i-1) + C(i)$
$S_2$: $B(i) = A(i+2) + C(i)$

endfor

Statements $S_1$ and $S_2$ are executed 50 times. We say $S_2(10)$ to mean the execution of $S_2$ when $i = 10$.

$S_1(1)$: $A(1) = B(0) + C(1)$
$S_2(1)$: $B(1) = A(3) + C(1)$
$S_1(2)$: $A(2) = B(1) + C(2)$
$S_2(2)$: $B(2) = A(4) + C(2)$
$S_1(3)$: $A(3) = B(2) + C(3)$
$S_2(3)$: $B(3) = A(5) + C(3)$

$\cdots \cdots \cdots$ 

$S_1(50)$: $A(50) = B(49) + C(50)$
$S_2(50)$: $B(50) = A(52) + C(50)$
Data flow dependence

Data written by some statement instance is later read by some statement instance, in the serial execution of the program. This is written as $S_1 \delta^f S_2$.

\[
\text{for } I = 1 \text{ to } 40 \text{ do }
\begin{align*}
S1: & \quad A(I+1) = \ldots \\
S2: & \quad \ldots = A(I-1)
\end{align*}
\text{endfor}
\]

Anti-dependence

Data read by some statement instance is later written by some statement instance, in the serial execution of the program. This is written as $S_1 \delta^a S_2$.

\[
\text{for } I = 1 \text{ to } 40 \text{ do }
\begin{align*}
S1: & \quad A(I-1) = \ldots \\
S2: & \quad \ldots = A(I+1)
\end{align*}
\text{enddo}
\]
Iteration space graph

Nested loops define an iteration space:

for \( i = 1 \) to 4 do
  for \( j = 1 \) to 4 do
    \( A(i,j) = B(i,j) + C(j) \)
  endfor
endfor

Sequential execution (traversal order):

Given two iterations \((i_1, j_1)\) and \((i_2, j_2)\) (with positive loop steps): we say \((i_1, j_1) \prec (i_2, j_2)\) if and only if either \(i_1 < i_2\) or \(i_1 = i_2\) \& \(j_1 < j_2\).

Any vector \((d_1, d_2)\) is positive, if \((0, 0) \prec (d_1, d_2)\) i.e., its first (leading) non-zero component is positive.
Dependences in loop nests

Loops of the form:

\[
\begin{align*}
\text{for } i_1 &= L_1 \text{ to } U_1 \text{ do} \\
\text{for } i_2 &= L_2 \text{ to } U_2 \text{ do} \\
\cdots \\
\text{for } i_n &= L_n \text{ to } U_n \text{ do} \\
&\text{BODY}(i_1, i_2, \ldots, i_n) \\
&\text{endfor} \\
\cdots \\
&\text{endfor} \\
\text{endfor}
\end{align*}
\]

There is a dependence in a loop nest if there are iterations \( \vec{I} = (i_1, i_2, \ldots, i_n) \) and \( \vec{J} = (j_1, j_2, \ldots, j_n) \) and some memory location \( M \) such that:

- \( \vec{I} < \vec{J} \)
- \( \text{BODY}(\vec{I}) \) and \( \text{BODY}(\vec{J}) \) reference \( M \)
- There is no intervening iteration \( \vec{K} \) that accesses \( M, \vec{I} < \vec{K} < \vec{J} \)

Distance and direction vectors

Assume a dependence from \( \text{BODY}((i_1, i_2, \ldots, i_n)) \) to \( \text{BODY}((j_1, j_2, \ldots, j_n)) \). The distance vector is \( \vec{d} = (j_1 - i_1, j_2 - i_2, \ldots, j_n - i_n) \).

Define the sign function \( sgn(x_1) \) of a scalar \( x_1 \):

\[
sgn(x_1) = \begin{cases} 
- & \text{if } x_1 < 0 \\
0 & \text{if } x_1 = 0 \\
+ & \text{if } x_1 > 0
\end{cases}
\]

The direction vector is \((sgn(d_1), sgn(d_2), \ldots, sgn(d_n))\) where \( d_k = j_k - i_k \) for \( k = 1, \ldots, n \).
Example of dependence vectors

for i = 1 to 5 do
  for j = i to 5 do
    S: \( A(i,j) = A(i,j-3) + A(i-2,j) + A(i-1,j+2) + A(i+1,j-1) \)
  endfor
endfor

<table>
<thead>
<tr>
<th>RHS ref.</th>
<th>Type</th>
<th>Distance vector</th>
<th>Direction vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A(i,j-3) )</td>
<td>flow</td>
<td>(0,3)</td>
<td>(0,+)</td>
</tr>
<tr>
<td>( A(i-2,j) )</td>
<td>flow</td>
<td>(2,0)</td>
<td>(+,0)</td>
</tr>
<tr>
<td>( A(i-1,j+2) )</td>
<td>flow</td>
<td>(1,-2)</td>
<td>(+,-)</td>
</tr>
<tr>
<td>( A(i+1,j-1) )</td>
<td>anti</td>
<td>(1,-1)</td>
<td>(+,-)</td>
</tr>
</tbody>
</table>

Validity of Loop interchange

Before interchange:

\[
\begin{align*}
&\text{for } i = 1 \text{ to } n \text{ do} \\
&\quad \text{for } j = 1 \text{ to } n \text{ do} \\
&\quad \quad \text{...} \\
&\quad \text{enddo} \\
&\text{enddo}
\end{align*}
\]

After interchange:

\[
\begin{align*}
&\text{for } j = 1 \text{ to } n \text{ do} \\
&\quad \text{for } i = 1 \text{ to } n \text{ do} \\
&\quad \quad \text{...} \\
&\quad \text{enddo} \\
&\text{enddo}
\end{align*}
\]

\((+,+)\) direction vector prevents interchange

Valid if all dependences are satisfied after interchange.

Geometric view: Source still executed before sink

Algebraic view: Permutated dep. vector is lex. +ve