Data Dependence, Parallelization, and Locality Enhancement

(courtesy of Tarek Abdelrahman, University of Toronto)

We define four types of data dependence.

- **Flow (true) dependence**: a statement $S_i$ precedes a statement $S_j$ in execution and $S_i$ computes a data value that $S_j$ uses.
  - It implies that $S_i$ must execute before $S_j$.
  $$S_i \Rightarrow S_j \quad (S_i \Rightarrow S_j \quad \text{and} \quad S_j \Rightarrow S_i)$$

- **Anti dependence**: a statement $S_i$ precedes a statement $S_j$ in execution and $S_i$ uses a data value that $S_j$ computes.
  - It implies that $S_i$ must be executed before $S_j$.
  $$S_i \Rightarrow S_j \quad (S_i \Rightarrow S_j \quad \text{and} \quad S_j \Rightarrow S_i)$$

- **Output dependence**: a statement $S_i$ precedes a statement $S_j$ in execution and $S_i$ computes a data value that $S_j$ also computes.
  - It implies that $S_i$ must be executed before $S_j$.
  $$S_i \Rightarrow S_j \quad (S_i \Rightarrow S_j \quad \text{and} \quad S_j \Rightarrow S_i)$$
**Data Dependence**

\[
\begin{align*}
S_1 &: A = 1.0 \\
S_2 &: B = A + 2.0 \\
S_3 &: A = C - D \\
S_4 &: A = B/C
\end{align*}
\]

We define four types of data dependence.

- **Input dependence**: a statement \(S_i\) precedes a statement \(S_j\) in execution and \(S_i\) uses a data value that \(S_j\) also uses.
- Does this imply that \(S_i\) must execute before \(S_j\)?

\[S_i \triangleright S_j \quad (S_i \triangleright S_j)\]

**Data Dependence (continued)**

- The dependence is said to **flow** from \(S_i\) to \(S_j\) because \(S_i\) precedes \(S_j\) in execution.
- \(S_i\) is said to be the **source** of the dependence. \(S_j\) is said to be the **sink** of the dependence.
- The only "true" dependence is flow dependence; it represents the flow of data in the program.
- The other types of dependence are caused by programming style; they may be eliminated by re-naming.

\[
\begin{align*}
S_1 &: A = 1.0 \\
S_2 &: B = A + 2.0 \\
S_3 &: A = C - D \\
S_4 &: A = B/C
\end{align*}
\]

**Value or Location?**

- There are two ways a dependence is defined: value-oriented or location-oriented.

\[
\begin{align*}
S_1 &: A = 1.0 \\
S_2 &: B = A + 2.0 \\
S_3 &: A = C - D \\
S_4 &: A = B/C
\end{align*}
\]
Example 1

```plaintext
do i = 2, 4
S1: \( a(i) = b(i) + c(i) \)
S2: \( d(i) = d(i) \)
end do
```

- There is an instance of \( S_1 \) that precedes an instance of \( S_2 \) in execution and \( S_1 \) produces data that \( S_2 \) consumes.
- \( S_1 \) is the source of the dependence; \( S_2 \) is the sink of the dependence.
- The number of iterations between source and sink (dependence distance) is 0. The dependence direction is =.

Example 2

```plaintext
do i = 2, 4
S1: \( a(i) = b(i) + c(i) \)
S2: \( d(i) = a(i-1) \)
end do
```

- There is an instance of \( S_1 \) that precedes an instance of \( S_2 \) in execution and \( S_1 \) produces data that \( S_2 \) consumes.
- \( S_1 \) is the source of the dependence; \( S_2 \) is the sink of the dependence.
- The dependence flows between instances of statements in different iterations (loop-carried dependence).
- The dependence distance is 1. The direction is positive (=).

Example 3

```plaintext
do i = 2, 4
S1: \( a(i) = b(i) + c(i) \)
S2: \( d(i) = a(i+1) \)
end do
```

- There is an instance of \( S_2 \) that precedes an instance of \( S_1 \) in execution and \( S_2 \) consumes data that \( S_1 \) produces.
- \( S_2 \) is the source of the dependence; \( S_1 \) is the sink of the dependence.
- The dependence is loop-carried.
- The dependence distance is 1.
  \( S_2 \prec S_1 \) or \( S_2 \succ S_1 \)

- Are you sure you know why it is \( S_2 \prec S_1 \) even though \( S_1 \) appears before \( S_2 \) in the code?

Example 4

```plaintext
do i = 2, 4
do j = 2, 4
S: \( a(j) = c(i) + c(j) \)
end do
end do
```

- An instance of \( S \) precedes another instance of \( S \) and \( S \) produces data that \( S \) consumes.
- \( S \) is both source and sink.
- The dependence is loop-carried.
- The dependence is (1,-1).
Problem Formulation

- Consider the following perfect nest of depth d:

\[
\begin{align*}
\text{do } i_1 &= L_1, U_1 \\
\text{do } i_2 &= L_2, U_2 \\
\text{enddo}
\end{align*}
\]

- Dependence will exist if there exists two iteration vectors \( i \) and \( j \) such that \( L \leq k \leq U \) and:

\[
\begin{align*}
\text{and } f_k(i) &= g_k(j) \\
\text{and } f_{k+1}(i) &= g_{k+1}(j) \\
\text{and } f_{k+n}(i) &= g_{k+n}(j)
\end{align*}
\]

- That is:

\[
\begin{align*}
f_k(i) - g_k(j) &= 0 \\
f_{k+1}(i) - g_{k+1}(j) &= 0 \\
f_{k+n}(i) - g_{k+n}(j) &= 0
\end{align*}
\]

Problem Formulation - Example

- Does there exist two iteration vectors \( i_1 \) and \( i_2 \), such that \( 2 \leq i_1 \leq i_2 \leq 4 \) and such that:

\[
\begin{align*}
i_1 &= i_2 + 1? \\
\text{Answer: yes; } i_2 &= 3 \text{ and } i_1 = 4.
\end{align*}
\]

- Hence, there is dependence!

- The dependence distance vector is \( i_2 - i_1 = 1 \).

- The dependence direction vector is \( \text{sign}(1) = \langle 1 \rangle \).

Problem Formulation - Example

- Does there exist two iteration vectors \( i_1 \) and \( i_2 \), such that \( 2 \leq i_1 \leq i_2 \leq 4 \) and such that:

\[
\begin{align*}
i_1 &= i_2 + 1? \\
\text{Answer: yes; } i_2 &= 3 \text{ and } i_1 = 4.
\end{align*}
\]

- Hence, there is dependence!

- The dependence distance vector is \( i_2 - i_1 = -1 \).

- The dependence direction vector is \( \text{sign}(-1) = \langle -1 \rangle \).

- Is this possible?
Problem Formulation - Example

\begin{verbatim}
do i = 1, 10
S1: a(2*i) = b(i) + c(i)
S2: d(i) = a(2*i+1)
end do
\end{verbatim}

- Does there exist two iteration vectors \( i_1 \) and \( i_2 \), such that 
  \( 1 \leq i_1 \leq i_2 \leq 10 \) and such that:
  \[ 2*i_1 = 2*i_2 + 1? \]
- Answer: no; \( 2*i_1 \) is even & \( 2*i_2+1 \) is odd.
- Hence, there is no dependence!

Dependence Testers

- Lamport's Test.
- GCD Test.
- Banerjee's Inequalities.
- Generalized GCD Test.
- Power Test.
- I-Test.
- Omega Test.
- Delta Test.
- Stanford Test.
- etc...

Problem Formulation

- Dependence testing is equivalent to an integer linear programming (ILP) problem of \( 2d \) variables & \( m+d \) constraint!
- An algorithm that determines if there exits two iteration vectors \( k \) and \( j \) that satisfies these constraints is called a dependence tester.
- The dependence distance vector is given by \( j - k \).
- The dependence direction vector is give by \( \text{sign}(j - k) \).
- Dependence testing is NP-complete!
- A dependence test that reports dependence only when there is dependence is said to be exact. Otherwise it is in-exact.
- A dependence test must be conservative; if the existence of dependence cannot be ascertained, dependence must be assumed.

Lamport's Test

- Lamport's Test is used when there is a single index variable in the subscript expressions, and when the coefficients of the index variable in both expressions are the same.

\[ A(\ldots, b * i + c_1, \ldots) = \ldots = A(\ldots, b * i + c_2, \ldots) \]

- The dependence problem: does there exist \( i_1 \) and \( i_2 \), such that \( L_i \leq i_1 \leq i_2 \leq U_i \) and such that
  \[ b*i_1 + c_1 = b*i_2 + c_2 \quad \text{or} \quad i_2 - i_1 = \frac{c_1 - c_2}{b} \]
- There is integer solution if and only if \( \frac{c_1 - c_2}{b} \) is integer.
- The dependence distance is \( d = \frac{c_1 - c_2}{b} \) if \( L_i \leq |d| \leq U_i \).
- \( d > 0 \) \( \Rightarrow \) true dependence.
- \( d = 0 \) \( \Rightarrow \) loop independent dependence.
- \( d < 0 \) \( \Rightarrow \) anti dependence.
Lamport's Test - Example

\[
\text{do } i = 1, n \\
\text{do } j = 1, n \\
S: a(i,j) = a(i-1,j+1) \\
\text{end do} \\
\text{end do}
\]

- \( i_1 = i_2 - 1 \)?
- \( j_1 = j_2 + 1 \)?

- \( b = 1; c_1 = 0; c_2 = -1 \)
- \( c_1 - c_2 = 1 \)

There is dependence. Distance (j) is -1.

- \( b = 1; c_1 = 0; c_2 = 1 \)
- \( c_1 - c_2 = -1 \)

There is dependence. Distance (i) is 1.

GCD Test

- Given the following equation:

\[
\sum_{i=1}^{n} a_i x_i = c
\]

\( a_i \)'s and c are integers

- an integer solution exists if and only if:

\[
gcd(a_1, a_2, \cdots, a_n) \text{ divides } c
\]

- Problems:
- ignores loop bounds.
- gives no information on distance or direction of dependence.
- often gcd(...) is 1 which always divides c, resulting in false dependences.

- Does there exist two iteration vectors \( i_1 \) and \( i_2 \), such that \( 1 \leq i_1 \leq i_2 \leq 10 \) and such that:

\[
2^*i_1 = 2^*i_2 - 1
\]

or

\[
2^*i_2 - 2^*i_1 = 1
\]

- There will be an integer solution if and only if gcd(2, -2) divides 1.
- This is not the case, and hence, there is no dependence!
**GCD Test Example**

```
do i = 1, 10
  S1: a(i) = b(i) * c(i)
  S2: d(i) = a(i-100)
end do
```

- Does there exist two iteration vectors $i_1$ and $i_2$, such that $1 \leq i_1 \leq i_2 \leq 10$ and such that:
  - $i_1 = i_2 - 100$?
  - or
  - $i_2 - i_1 = 100$?
- There will be an integer solution if and only if $\gcd(1, -1)$ divides 100.
- This is the case, and hence, there is dependence! Or is there?

**Dependence Testing Complications**

- Unknown loop bounds.
  ```
  do i = 1, N
    S: a(i) = a(i+10)
  end do
  ```
  What is the relationship between $N$ and 10?

- Triangular loops.
  ```
  do i = 1, N
    do j = 1, i-1
      S: a(i,j) = a(j,i)
    end do
  end do
  ```
  Must impose $j < i$ as an additional constraint.

**More Complications**

- User variables.
  ```
  do i = 1, 10
    S1: a(i) = a(i+k)
    end do
  ```
  Same problem as unknown loop bounds, but occur due to some loop transformations (e.g., normalization).

  ```
  do i = L, H
    S1: a(i) = a(i-1)
    end do
  ```

```
  $$j = N-1$$
  do i = 1, N
    S1: a(i) = a(N-i)
  end do

  sum = 0
  do i = 1, N
    S1: sum += a(i)
  end do
```

- Scalars.
  ```
  do i = 1, N
    x = a(i)
  end do
  ```

  ```
  do i = 1, N
    x = a(N-i)
  end do
  ```

  ```
  sum = 0
  do i = 1, N
    sum = sum + a(i)
  end do
  ```

  ```
  sum = sum + a(i)
  ```
Serious Complications

- Aliases.
  - Equivalence Statements in Fortran:
    ```fortran
    real a(10,10), b(10)
    makes b the same as the first column of a.
    ```
  - Common blocks: Fortran's way of having shared/global variables.
    ```fortran
    common /shared/a,b,c
    ```

Loop Parallelization

- A dependence is said to be carried by a loop if the loop is the outmost loop whose removal eliminates the dependence. If a dependence is not carried by the loop, it is loop-independent.
  ```fortran
  do i = 2, n-1
    do j = 2, m-1
      a(i, j) = ...
      c(i, j) = ...
      b(i, j) = ...
      c(i, j) = ...
      end do
    end do
  end do
  ```
Loop Parallelization

A dependence is said to be carried by a loop if the loop is the outmost loop whose removal eliminates the dependence. If a dependence is not carried by the loop, it is loop-independent.

\[
\begin{align*}
\text{do } i &= 2, n-1 \\
\text{do } j &= 2, m-1 \\
\quad a(i, j) &= \cdots = a(i, j) \\
\quad b(i, j) &= \cdots = b(i, j-1) \\
\delta_j &= c(i, j) = \cdots = c(i-1, j) \\
\end{align*}
\]

end do
end do

Loop Parallelization - Example

The iterations of a loop may be executed in parallel with one another if and only if no dependences are carried by the loop!

\[
\begin{align*}
\text{do } i &= 2, n-1 \\
\text{do } j &= 2, m-1 \\
\quad a(i, j) &= \cdots = a(i, j) \\
\delta_j &= b(i, j) = \cdots = b(i, j-1) \\
\delta_j &= c(i, j) = \cdots = c(i-1, j) \\
\end{align*}
\]

end do
end do

• Iterations of loop j must be executed sequentially, but the iterations of loop i may be executed in parallel.
• Outer loop parallelism.
Loop Parallelization - Example

1. Iterations of loop i must be executed sequentially, but the iterations of loop j may be executed in parallel.
2. Inner loop parallelism.

Loop Interchange

Loop interchange changes the order of the loops to improve the spatial locality of a program.

- Loop Parallelization - Example

- Loop Interchange
Loop Interchange

- Loop interchange can improve the granularity of parallelism!

\[
\begin{align*}
    &\text{do } i = 1, n \\
    &\text{do } j = 1, n \\
    &a(i,j) = b(i,j) \\
    &c(i,j) = a(i-1,j) \\
    &\text{end do} \\
    &\text{end do} \\
\end{align*}
\]

\[
\begin{align*}
    &\text{do } j = 1, n \\
    &\text{do } i = 1, n \\
    &a(i,j) = b(i,j) \\
    &c(i,j) = a(i-1,j) \\
    &\text{end do} \\
    &\text{end do} \\
\end{align*}
\]

- When is loop interchange legal?
Loop Interchange

When is loop interchange legal? when the "interchanged" dependences remain lexicographically positive!

Loop Blocking (Loop Tiling)
Exploits temporal locality in a loop nest.

do ic = 1, n, B
do jc = 1, n, B
do t = 1, T
  do i = 1, B
    do j = 1, B
      \( a(i, j) \) ...
    end do
  end do
end do
B: Block size
Loop Blocking (Loop Tiling)
Exploits temporal locality in a loop nest.

- When is loop blocking legal?