Lecture 17:
Array Dependence Analysis & Parallelization

I. Data Dependence
II. Dependence Testing: Formulation
III. Dependence Testers
IV. Loop Parallelization
V. Loop Interchange

[ALSU 11.6]
I. Data Dependence

\[ S_1 : \quad A = 1.0 \]
\[ S_2 : \quad B = A + 2.0 \]
\[ S_3 : \quad A = C - D \]
\[ \vdots \]
\[ S_4 : \quad A = B/C \]

We define four types of data dependence.

1. **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.
   - Implies that \( S_i \) must execute before \( S_j \).

   \[ S_i \delta^+ S_j \quad (S_1 \delta^+ S_2 \quad \text{and} \quad S_2 \delta^+ S_4) \]
Data Dependence

\[ S_1 : \quad A = 1.0 \]
\[ S_2 : \quad B = A + 2.0 \]
\[ S_3 : \quad A = C - D \]
\[ \vdots \]
\[ S_4 : \quad A = B/C \]

We define four types of data dependence.

2. **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 \( S_j \).

\[ S_i \delta^a S_j \quad (S_2 \delta^a S_3) \]
Data Dependence

\[ S_1 : A = 1.0 \]
\[ S_2 : B = A + 2.0 \]
\[ S_3 : A = C - D \]
\[ \vdots \]
\[ S_4 : A = B/C \]

We define four types of data dependence.

3. **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 \( S_j \).

\[ S_i \delta S_j \quad (S_1 \delta S_3 \quad \text{and} \quad S_3 \delta S_4) \]
4. 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 $S_j$? **no**

\[ S_i \delta^I S_j \quad (S_3 \delta^I S_4) \]
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.

\[
\begin{align*}
S_1 & : A = 1.0 \\
S_2 & : B = A + 2.0 \\
S_3 & : A1 = C - D \\
& \quad \vdots \\
S_4 & : A2 = B/C
\end{align*}
\]
Data Dependence (continued)

- Data dependence in a program may be represented using a dependence graph $G=(V,E)$, where the nodes $V$ represent statements in the program and the directed edges $E$ represent dependence relations.

- $S_1 : A = 1.0$
- $S_2 : B = A + 2.0$
- $S_3 : A = C - D$
- $\vdots$
- $S_4 : A = B/C$
Example 1

for $i = 2$ to $4$

$S_1$: $a[i] = b[i] + c[i]$;

$S_2$: $d[i] = a[i]$

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 the same iteration (loop-independent dependence).

The number of iterations between source and sink (dependence distance) is 0. The dependence direction is $= \delta^t$.

$S_1 \delta^t S_2$ or $S_1 \delta^t_0 S_2$
Example 2

\[
\begin{align*}
\text{do } & i = 2, 4 \\
S_1: & \quad a(i) = b(i) + c(i) \\
S_2: & \quad d(i) = a(i-1) \\
\text{end do}
\end{align*}
\]

- 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 (\( < \)).

\[
S_1 \delta^t < S_2 \quad \text{or} \quad S_1 \delta^t_1 S_2
\]
Example 3

\[
\begin{align*}
  \text{do } & i = 2, 4 \\
  S_1: & \ a(i) = b(i) + c(i) \\
  S_2: & \ d(i) = a(i+1) \\
  \text{end do}
\end{align*}
\]

- 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.

\[
S_2 \triangleright^a S_1 \quad \text{or} \quad S_2 \triangleleft^a S_1
\]

- Are you sure you know why it is \( S_2 \triangleright^a S_1 \) even though \( S_1 \) appears before \( S_2 \) in the code?
Example 4: 2D Iteration Space

do i = 2, 4
  do j = 2, 4
    S: \( a(i,j) = a(i-1,j+1) \)
  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 distance is \((1,-1)\).

\[ S_{\delta}^t \quad \text{or} \quad S_{\delta}^{\dagger} \]

\[ \delta_{(1,-1)} \]
II. Dependence Testing: Formulation

- Consider the following perfect nest of depth $d$:

\[ \begin{align*}
    \text{do } I_1 &= L_1, U_1 \\
    \text{do } I_2 &= L_2, U_2 \\
    &\vdots \\
    \text{do } I_d &= L_d, U_d \\
    \quad a(f_1(I), f_2(I), \ldots, f_m(I)) &= \ldots \\
    \quad \ldots &= a(g_1(I), g_2(I), \ldots, g_m(I)) \\
    \text{enddo} \\
    &\vdots \\
    \text{enddo} \\
    \text{enddo}
\end{align*} \]

$\bar{I} = (I_1, I_2, \ldots, I_d)$
$\bar{L} = (L_1, L_2, \ldots, L_d)$
$\bar{U} = (U_1, U_2, \ldots, U_d)$
$\bar{L} \leq \bar{U}$

“perfect” means step=1
Problem Formulation

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

\[
\begin{align*}
&f_1(\vec{k}) = g_1(\vec{j}) \\
&f_2(\vec{k}) = g_2(\vec{j}) \\
&\vdots \\
&f_m(\vec{k}) = g_m(\vec{j})
\end{align*}
\]

- That is:

\[
\begin{align*}
&f_1(\vec{k}) - g_1(\vec{j}) = 0 \\
&f_2(\vec{k}) - g_2(\vec{j}) = 0 \\
&\vdots \\
&f_m(\vec{k}) - g_m(\vec{j}) = 0
\end{align*}
\]
Problem Formulation - Example

\[
\begin{align*}
d & \text{do } i = 2, 4 \\
S_1 & : \quad a(i) = b(i) + c(i) \\
S_2 & : \quad d(i) = a(i-1) \\
\text{end do}
\end{align*}
\]

- 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:

\[i_1 = i_2 - 1?\]

- Answer: yes; \(i_1=2\) & \(i_2=3\) and \(i_1=3\) & \(i_2=4\).

- Hence, there is dependence!

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

- The dependence direction vector is \(\text{sign}(1) = <\).
Problem Formulation - Example

do i = 2, 4
  \( S_1: \quad a(i) = b(i) + c(i) \)
  \( S_2: \quad d(i) = a(i+1) \)
end do

- 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:

  \[ i_1 = i_2 + 1? \]

- Answer: yes; \( i_1 = 3 \) & \( i_2 = 2 \) and \( i_1 = 4 \) & \( i_2 = 3 \). (But, but!).

- Hence, there is dependence!

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

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

- Is this possible? As an antidependence, not a true dependence
Problem Formulation - Example

\[
\begin{align*}
\text{do } i &= 1, 10 \\
S_1: & \quad a(2*i) = b(i) + c(i) \\
S_2: & \quad d(i) = a(2*i+1) \\
\text{end do}
\end{align*}
\]

- 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!
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 exists two iteration vectors $\vec{k}$ and $\vec{j}$ that satisfies these constraints is called a dependence tester.

\[
\begin{align*}
\text{do } I_1 &= L_1, U_1 \\
\text{do } I_2 &= L_2, U_2 \\
\ldots \nonumber \\
\text{do } I_d &= L_d, U_d \\
a(f_1(I), f_2(I), \ldots, f_m(I)) &= \ldots \\
\ldots &= a(g_1(I), g_2(I), \ldots, g_m(I)) \\
\text{enddo} \\
\ldots \\
\text{enddo} \\
\text{enddo}
\end{align*}
\]
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 exists two iteration vectors \( \vec{k} \) and \( \vec{j} \) that satisfies these constraints is called a dependence tester.

- The dependence distance vector is given by \( \vec{j} - \vec{k} \).

- The dependence direction vector is given by \( \text{sign} ( \vec{j} - \vec{k} ) \).

- 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.
III. Dependence Testers

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

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

• 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 \( L_i \leq |d| \leq U_i \).

• \( d > 0 \Rightarrow true \ dependency. \)
  \( d = 0 \Rightarrow loop \ independent \ dependence. \)
  \( d < 0 \Rightarrow anti \ dependency. \)
Lamport’s Test - Example

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

- \( i_1 = i_2 - 1? \)
  - \( b = 1; \ c_1 = 0; \ c_2 = -1 \)
  - \( \frac{c_1 - c_2}{b} = 1 \)
  - There is dependence.
  - Distance (i) is 1.

- \( j_1 = j_2 + 1? \)
  - \( b = 1; \ c_1 = 0; \ c_2 = 1 \)
  - \( \frac{c_1 - c_2}{b} = -1 \)
  - There is dependence.
  - Distance (j) is -1.

\[
S \delta^+_t (1,-1) S \quad \text{or} \quad S \delta^+_t (<>>)_S
\]
Lamport’s Test - Example

\[
\begin{align*}
&\text{do } i = 1, n \\
&\text{do } j = 1, n \\
&S: \quad a(i,2* j) = a(i-1,2* j+1) \\
&\text{end do} \\
&\text{end do} \\
\text{• } i_1 = i_2 - 1? \\
&b = 1; c_1 = 0; c_2 = -1 \\
&\frac{c_1 - c_2}{b} = 1 \\
&\text{There is dependence.} \\
&\text{Distance (i) is 1.}
\end{align*}
\]

\[
\begin{align*}
&\text{• } 2* j_1 = 2* j_2 + 1? \\
&b = 2; c_1 = 0; c_2 = 1 \\
&\frac{c_1 - c_2}{b} = \frac{-1}{2} \\
&\text{There is no dependence.}
\end{align*}
\]
**GCD Test**

- **Given the following equation:**

\[ \sum_{i=1}^{n} a_i x_i = c \] where \( a_i \) and \( c \) are integers

an integer solution exists if and only if:

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

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

\[
\text{do } i = 1, 10 \\
S_1: \quad a(2i) = b(i) + c(i) \\
S_2: \quad d(i) = a(2i-1) \\
\text{end do}
\]

- 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:

\[
2i_1 = 2i_2 - 1?
\]

or

\[
2i_2 - 2i_1 = 1?
\]

- 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

\begin{verbatim}
do i = 1, 10
  \textbf{S}_1: \quad a(i) = b(i) + c(i)
  \textbf{S}_2: \quad d(i) = a(i-100)
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:
  \[ i_1 = i_2 - 100? \]
  or
  \[ i_2 - i_1 = 100? \]

- There will be an integer solution if and only if \(\text{gcd}(1,-1)\) divides 100.

- This is the case, and hence, there is dependence! Or is there?

No: check loop bounds. Shows a limitation of GCD.
Dependence Testing: Complications

- Unknown loop bounds:

  \[
  \text{do } i = 1, N \\
  S_1: \quad a(i) = a(i+10) \\
  \text{end do}
  \]

  What is the relationship between \(N\) and 10?

- Triangular loops:

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

  Must impose \(j < i\) as an additional constraint.
More Complications

- User variables:

\[
\text{do } i = 1, 10 \\
S_1: \quad a(i) = a(i+k) \\
\text{end do}
\]

Same problem as unknown loop bounds, but occur due to some loop transformations (e.g., loop bounds normalization).

\[
\text{do } i = L, H \\
S_1: \quad a(i) = a(i-1) \\
\text{end do}
\]

\[
\downarrow
\]

\[
\text{do } i = 1, H-L \\
S_1: \quad a(i+L) = a(i+L-1) \\
\text{end do}
\]
More Complications

- Scalars:

\[
\begin{align*}
\text{do } i &= 1, N \\
S_1 &: \quad x = a(i) \\
S_2 &: \quad b(i) = x \\
\end{align*}
\]

\[
\begin{align*}
\text{do } i &= 1, N \\
S_1 &: \quad x(i) = a(i) \\
S_2 &: \quad b(i) = x(i) \\
\end{align*}
\]

\[
\begin{align*}
\text{j} &= N-1 \\
\text{do } i &= 1, N \\
S_1 &: \quad a(i) = a(j) \\
S_2 &: \quad j = j - 1 \\
\end{align*}
\]

\[
\begin{align*}
\text{do } i &= 1, N \\
S_1 &: \quad a(i) = a(N-i) \\
\end{align*}
\]

\[
\begin{align*}
\text{sum} &= 0 \\
\text{do } i &= 1, N \\
S_1 &: \quad \text{sum} = \text{sum} + a(i) \\
\end{align*}
\]

\[
\begin{align*}
\text{do } i &= 1, N \\
S_1 &: \quad \text{sum}(i) = a(i) \\
\end{align*}
\]

\[
\begin{align*}
\text{sum} &= \text{sum} + \text{sum}(i) \quad \text{i} = 1, N
\end{align*}
\]
IV. Loop Parallelization

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

```plaintext
do i = 2, n-1
    do j = 2, m-1
        a(i, j) = ...
        ... = a(i, j)
        b(i, j) = ...
        ... = b(i, j-1)
        c(i, j) = ...
        ... = c(i-1, j)
    end do
end do
```
Loop Parallelization

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

\begin{align*}
do i = 2, n-1 \\
do j = 2, m-1 \\
\quad a(i, j) &= \ldots \\
\delta^+_{=,=} &\quad \ldots = a(i, j) \\
\quad b(i, j) &= \ldots \\
\delta^+_{=,<} &\quad \ldots = b(i, j-1) \\
\quad c(i, j) &= \ldots \\
\delta^+_{<,=} &\quad \ldots = c(i-1, j) \\
&\text{end do} \\
&\text{end do}
\end{align*}

- Outermost loop with a non “=“ direction carries dependence!
Loop Parallelization

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

- 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

- Iterations of loop i must be executed sequentially, but the iterations of loop j may be executed in parallel.

- Inner loop parallelism. (Vectorization, SIMD)
Loop Parallelization - Example

- Iterations of loop $i$ must be executed sequentially, but the iterations of loop $j$ may be executed in parallel. Why?

- Inner loop parallelism.
V. Loop Interchange

Improves spatial locality

\[
\begin{align*}
&\text{for } i = 0 \text{ to } N-1 \\
&\quad \text{for } j = 0 \text{ to } N-1 \\
&\quad A[j][i] = i \times j;
\end{align*}
\]

Assume row major order, N large, 4 elements per cache line

Hit
Miss
**Loop Interchange**

*Can improve the granularity of parallelism*

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

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

\(\delta_{<,=}^{+}\) \hspace{1cm} \(\delta_{=,<}^{+}\)

**Inner loop** \hspace{1cm} **Outer loop**
Loop Interchange

- When is loop interchange legal?
Loop Interchange

Focus only on true dependences

- When is loop interchange legal?
Loop Interchange

- When is loop interchange legal?
When is loop interchange legal?  
when the “interchanged” 
dependences remain lexiographically positive!
Today’s Class: Array Dependence Analysis & Parallelization

I. Data Dependence
II. Dependence Testing: Formulation
III. Dependence Testers
IV. Loop Parallelization
V. Loop Interchange

Friday’s Class

• Student Presentations (1/3)