Class Announcements

Here is where we are:

- Second project due Monday, April 24
- Third project has been posted; due Monday, May 1
- Sixth homework will be posted later today or tomorrow; due Tuesday, April 25
- Seventh homework will be posted next week; due Monday, May 1
- Final (third midterm) exam on Thursday, May 4, noon - 3:00pm timeslot. Location: most likely this room.

Any CONFLICTS with other classes?
Dependence — Overview

**Definition** — There is a data dependence from statement $S_1$ to statement $S_2$ ($S_1\delta S_2$) if

1. Both statements access the same memory location, and
2. There is a run–time execution path from $S_1$ to $S_2$.

**Data dependence classification**

“$S_2$ depends on $S_1$” — $S_1\delta S_2$

**true (flow) dependence**
occurs when $S_1$ writes a memory location that $S_2$ later reads

**anti dependence**
occurs when $S_1$ reads a memory location that $S_2$ later writes

**output dependence**
occurs when $S_1$ writes a memory location that $S_2$ later writes

**input dependence**
occurs when $S_1$ reads a memory location that $S_2$ later reads.

Note: Input dependences do not restrict statement (*load/store*) order!
Dependence Analysis

Question

Do two variable references never/maybe/always access the same memory location?

Benefits

- improves alias analysis
- enables loop transformations

Motivation

- classic optimizations
- instruction scheduling
- data locality (register/cache reuse)
- vectorization, parallelization

Obstacles

- array references
- pointer references
A **loop-independent** dependence exists regardless of the loop structure. The source and sink of the dependence occur on the same loop iteration.

A **loop-carried** dependence is induced by the iterations of a loop. The source and sink of the dependence occur on different loop iterations.

*Loop-carried dependences can inhibit parallelization and loop transformations*
Dependence Testing

Given

\[
\begin{align*}
\text{do } i_1 &= L_1, U_1 \\
&\quad \ldots \\
&\quad \text{do } i_n = L_n, U_n \\
S_1 &= A(f_1(i_1, \ldots, i_n), \ldots, f_m(i_1, \ldots, i_n)) = \ldots \\
S_2 &= \ldots = A(g_1(i_1, \ldots, i_n), \ldots, g_m(i_1, \ldots, i_n))
\end{align*}
\]

A dependence between statement \(S_1\) and \(S_2\), denoted \(S_1 \delta S_2\), indicates that \(S_1\), the source, must be executed before \(S_2\), the sink on some iteration of the nest.

Let \(\alpha \& \beta\) be a vector of \(n\) integers within the ranges of the lower and upper bounds of the \(n\) loops.

Does \(\exists \alpha \leq \beta\), s.t.

\[
f_k(\alpha) = g_k(\beta) \quad \forall k, \ 1 \leq k \leq m?
\]
Iteration Space

\[
\begin{align*}
&\text{do } I = 1, 5 \\
&\quad \text{do } J = I, 6 \\
&\quad \quad \quad \ldots \\
&\quad \quad \text{enddo} \\
&\quad \text{enddo} \\
&1 \leq I \leq 5 \\
&I \leq J \leq 6 \\
\end{align*}
\]

- lexicographical (sequential) order for the above iteration space is

\[
(1,1), (1,2), \ldots, (1,6) \\
(2,2), (2,3), \ldots, (2,6) \\
\ldots \\
(5,5), (5,6)
\]

- given \( I = (i_1, \ldots, i_n) \) and \( I' = (i'_1, \ldots, i'_n) \),

\[
I < I' \ \text{iff} \\
(i_1, i_2, \ldots, i_k) = (i'_1, i'_2, \ldots, i'_k) \ \& \ i_{k+1} < i'_{k+1}
\]
Distance & Direction Vectors

Distance Vector = number of iterations between accesses to the same location

Direction Vector = direction in iteration space (=, <, >)

<table>
<thead>
<tr>
<th></th>
<th>distance vector</th>
<th>direction vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1\delta S_1$</td>
<td>$(0, 1)$</td>
<td>$(=, &lt;)$</td>
</tr>
<tr>
<td>$S_2\delta S_2$</td>
<td>$(1, 1)$</td>
<td>$(&lt;, &lt;)$</td>
</tr>
<tr>
<td>$S_3\delta S_3$</td>
<td>$(1, -1)$</td>
<td>$(&lt;, &gt;)$</td>
</tr>
</tbody>
</table>
Approaches to Dependence Testing

- can we solve this problem exactly?
- what is conservative in this framework?
- restrict the problem to consider index and bound expressions that are linear functions

\[ \Rightarrow \text{solving general system of linear equations in integers is NP-hard} \]

Solution Methods

- inexact methods
  - Greatest Common Divisor (GCD)
  - Banerjee’s inequalities
- cascade of exact, efficient tests
  (fall back on inexact methods if needed)
  - Rice (see posted PLDI’91 paper)
  - Stanford
- exact general tests
  (integer programming)
Dependence Testing

SIV - Single Induction Variable Test

1. Single loop nest with constant lower (LB) and upper (UB) bounds, and step 1
   
   for i = LB, UB, 1
   
   ... 
   
   endfor

   The loop bounds define the iteration space for loop induction variable i.

2. Two array references with array subscript (index) expressions of the form (true dependence)

   for i = LB, UB, 1
   
   R1: X(a*i + c1) = ... \ write
   R2: ... X(a*i + c2) ... \ read
   
   endfor

   where a, c1, and c2 are integer constants, R1 and R2 are references to the same array, i is the loop induction variable, and a \( \neq 0 \).

Question: Is there a true dependence between R1 and R2?
Dependence Testing

There is a dependence between R1 and R2 iff

$$\exists i, i' : i \leq i' \text{ and } (a \times i + c_1) = (a \times i' + c_2)$$

where $i$ and $i'$ are two iterations in the iteration space of the loop. This means that in both iterations, the same element of array $X$ would be accessed.

So let’s just solve the equation:

$$(a \times i + c_1) = (a \times i' + c_2) \iff$$

$$\frac{c_1 - c_2}{a} = i' - i = \Delta d$$

There is a dependence with distance $\Delta d$ iff

1. $\Delta d$ is an integer value and
2. UB - LB $\geq \Delta d \geq 0$
Dependence Testing Examples

1. for i = LB, UB, 1
   R1: X(i) = ... \ write
   R2: ... X(i - 2) ... \ read
   endfor

   a=1, c_1=0, c_2=-2 ⇒ Δd = 2 (dependence)

2. for i = LB, UB, 1
   R1: X(2*i) = ... \ write
   R2: ... X(2*i - 1) ... \ read
   endfor

   a=2, c_1=0, c_2=-1 ⇒ Δd = \frac{1}{2} (no dependence)

Assume R1 executes before R2.

[Classification of dependences:]

- R1 is write, R2 is read ⇒ true dependence
- R1 is read, R2 is write ⇒ anti dependence
- R1 is write, R2 is write ⇒ output dependence
Next Lecture

Things to do:

- Please see our OpenMP tutorial on our website
- More on dependence testing and loop optimizations