REMINDERS

• Homework 8 due on Tuesday, December 8 (last lecture); no late submission.

• Final exam: Thursday, December 17, 4:00pm - 7:00pm, BRR 1095; accumulative, closed books, closed notes. For students with \texttt{cs211} or \texttt{cs336} conflicts, you will have to be in Tillett Hall 258 (Livingston campus) at noon, Thursday, December 17. The conflicting exams will be given back to back, so no need to move to another exam site.

• Project 3 new deadline: Saturday, December 12, at 11:59pm.
Dependence Testing

Given

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

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.

\[
f_k(\alpha) = g_k(\beta) \quad \forall k, 1 \leq k \leq m
\]
Which Loops are Parallel?

\[
\begin{align*}
S_1 & \quad A(I,J) = A(I,J-1) \\
S_2 & \quad A(I,J) = A(I-1,J-1) \\
S_3 & \quad B(I,J) = B(I-1,J+1)
\end{align*}
\]

- a dependence \( D = (d_1, \ldots, d_k) \) is carried at level \( i \), if \( d_i \) is the first nonzero element of the distance/direction vector

- a loop \( l_i \) is parallel, if \( \forall \) a dependence \( D_j \) carried at level \( i \)

\[
\begin{array}{c|c|c}
\text{distance vector} & \text{direction vector} \\
\hline
\forall D_j & d_1, \ldots, d_{i-1} > 0 & d_1, \ldots, d_{i-1} = "<" \\
\text{OR} & d_1, \ldots, d_i = 0 & d_1, \ldots, d_i = "="
\end{array}
\]
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

⇒ 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

   \[ \text{for } i = \text{LB, UB, 1} \]
   \[ \ldots \]
   \[ \text{endfor} \]

   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)

   \[ \text{for } i = \text{LB, UB, 1} \]
   \[ \text{R1: } X(a*i + c1) = \ldots \quad \text{\\ write} \]
   \[ \text{R2: } \ldots X(a*i + c2) \ldots \quad \text{\\ read} \]
   \[ \text{endfor} \]

   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. \(\text{UB} - \text{LB} \geq \Delta d \geq 0\)
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 \Rightarrow \Delta d = 2 \text{ (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 \Rightarrow \Delta d = \frac{1}{2} \text{ (no dependence)}

Assume R1 executes before R2.

Classification of dependences:

- R1 is write, R2 is read \Rightarrow \text{true dependence}
- R1 is read, R2 is write \Rightarrow \text{anti dependence}
- R1 is write, R2 is write \Rightarrow \text{output dependence}
Dependence Testing

ZIV - Zero Induction Variable Test

Two array references with array subscript (index) expressions of the form of a constant:

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

where \( c_1 \) and \( c_2 \) are integer constants, and R1 and R2 are references to the same array.

There is a dependence between R1 and R2 iff

\[ c_1 = c_2 = c. \]

What is the dependence distance \( \Delta d \)?

Since every iteration \( i \) writes \( X(c) \), and every iteration \( i' \) reads \( X(c) \), there is no fixed distance \( \Delta d \). In fact, both references have true, anti, and output dependences:

\[ \Delta d \in \{0, \ldots UB - LB\} \text{ for true} \]
\[ \Delta d \in \{1, \ldots UB - LB\} \text{ for anti and output} \]
A Simple Vectorizing Compiler

How to vectorize the following loops?

for (i=2; i<100; i++) {
    S1: a[i] = b[i+1] + 1;
    S2: b[i] = a[i] + 5;
}

for (i=2; i<100; i++) {
    S1: a[i] = b[i-1] + a[i-1] + 3;
    S2: b[i] = a[i+1] + 5;
}

Simple vectorizer assumptions:

1. singly-nested loops
2. constant upper and lower bounds, step is always 1
3. body is sequence of assignment statements to array variables
4. simple array index expressions of induction variable (i +/- c or c); can use ZIV or SIV test
5. no function calls
A Simple Vectorizing Source-to-Source Compiler

SKETCH OF BASIC ALGORITHM

Here is a basic vectorization algorithm based on a statement-level dependence graph:

1. Construct statement-level dependence graph considering true, anti, and output dependences; in the final dependence graph, the type of the dependence is not important any more

2. Detect strongly connected components (SCC) over the dependence graph; represent SCC as summary nodes; walk resulting graph in topological order; For each visited node do

   (a) if SCC has more than one statement in it, distribute loop with statements of SCC as its body, and keep the code sequential

   (b) if SCC is a single statement and has no dependence cycle, distribute loop around it and generate vector code; otherwise, mark distributed loop sequential.
Next Lecture

Things to do:

• Loop transformations
• More on automatic vectorization
• Work on Project 3!