INFORMATION and REMINDERS

• Homework 8 due on Monday, May 1, 11:59pm; sample solution will be available May 2.

• Project 3
  – Deadline: Wednesday, May 3, 11:59pm
  – Required minimum performance improvements will be posted later.

• Review session: Wednesday, May 3, 11:00am - noon, CoRE 301

• FINAL EXAM
  1. Room assignments will be posted on our website.
  2. **Monday, May 8, 4:00-7:00pm**, College Ave. Campus
  3. **CONFLICTS?** by May 1.


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?
There is a dependence between R1 and R2 \textit{iff}

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

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 \( \Delta d \) \textit{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: $X(i) = \ldots$ \ \ write
   
   R2: $\ldots X(i - 2) \ldots$ \ \ read
   
   endfor

   $a=1, c_1=0, c_2=-2 \Rightarrow \Delta d = 2$ (dependence)

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

   $a=2, c_1=0, c_2=-1 \Rightarrow \Delta d = \frac{1}{2}$ (no dependence)

Assume R1 executes before R2.

Classification of dependences:

- R1 is write, R2 is read $\Rightarrow$ true dependence
- R1 is read, R2 is write $\Rightarrow$ anti dependence
- R1 is write, R2 is write $\Rightarrow$ output dependence
Dependence Testing

ZIV - Zero Induction Variable Test

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

\[
\text{for } i = \text{LB, UB, 1}
\]
\[
\text{R1: } X(c1) = \ldots \text{ \\\
write}
\]
\[
\text{R2: } \ldots X(c2) \ldots \text{ \\\nread}
\]
\text{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 \textit{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?

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

```c
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.
Loop Transformations

Goal

• modify execution order of loop iterations
• preserve data dependence constraints

Motivation

• data locality
  (increase reuse of registers, cache)

• parallelism
  (eliminate loop-carried deps, incr granularity)

Taxonomy

• loop interchange
  (change order of loops in nest)

• loop fusion
  (merge bodies of adjacent loops)

• loop distribution
  (split body of loop into adjacent loops)

• strip-mine and interchange (tiling, blocking)
  (split loop into nested loops, then interchange)
Loop Interchange

\[
\begin{align*}
&\text{do } I = 1, N \\
&\text{do } J = 1, N \\
&S_1 \quad A(I,J) = A(I,J-1) \\
&S_2 \quad B(I,J) = B(I-1,J-1) \\
&\text{enddo} \\
&\text{enddo}
\end{align*}
\]

\[\Rightarrow \text{loop interchange} \Rightarrow\]

\[
\begin{align*}
&\text{do } J = 1, N \\
&\text{do } I = 1, N \\
&S_1 \quad A(I,J) = A(I,J-1) \\
&S_2 \quad B(I,J) = B(I-1,J-1) \\
&\text{enddo} \\
&\text{enddo}
\end{align*}
\]

Loop interchange is safe \[\text{iff}\]

- it does not create a lexicographically negative direction vector \[(1,-1) \rightarrow (-1,1)\]

\[\Rightarrow \text{Benefits}\]

- may expose parallel loops, incr granularity
- reordering iterations may improve reuse
Loop Fusion

\[
\begin{align*}
\text{do } i &= 2, \text{ N} \\
S_1 &\quad A(i) = B(i) \\
\text{do } i &= 2, \text{ N} \\
S_2 &\quad B(i) = A(i-1)
\end{align*}
\]

\[\Rightarrow \text{ loop fusion } \Rightarrow\]

\[
\begin{align*}
\text{do } i &= 2, \text{ N} \\
S_1 &\quad A(i) = B(i) \\
S_2 &\quad B(i) = A(i-1)
\end{align*}
\]

Loop fusion is safe iff

• no loop-independent dependence between nests is converted to a backward loop-carried dep

(would fusion be safe if \(S_2\) referenced \(a(i+1)\)?)

⇒ Benefits

○ reduces loop overhead

○ improves reuse between loop nests

○ increases granularity of parallel loop
Loop Distribution

\[
\text{do } i = 2, N \\
S_1 \quad A(i) = B(i) \\
S_2 \quad B(i) = A(i-1)
\]

\[\Rightarrow \text{ loop distribution } \Rightarrow \]

\[
\text{do } i = 2, N \\
S_1 \quad A(i) = B(i) \\
\text{do } i = 2, N \\
S_2 \quad B(i) = A(i-1)
\]

**Loop distribution** is safe *iff*

- statements involved in a cycle of true deps *(recurrence)* remain in the same loop, and
- if $\exists$ a dependence between two statements placed in different loops, it must be forward

$\Rightarrow$ **Benefits**

- necessary for vectorization
- may enable partial/full parallelization
- may enable other loop transformations
- may reduce register/cache pressure
That’s it!

- Please look for final exam room assignments on our web site.
- See you at the review session on Wednesday.