INFORMATION and REMINDERS

- Homework 8 has been posted. Due Wednesday, December 13 at 11:59pm.

- Third programming has been posted. Due Friday, December 15, 11:59pm.

- Midterm sample solutions are available. Midterm grade challenge deadline: December 11.

- Final exam: Wednesday, December 20, 4:00-7:00pm, in our classroom.

DO YOU HAVE A CONFLICT? Need to know by Wednesday, December 13.

http://sasundergrad.rutgers.edu/forms/final-exam-conflict

- More than two (2) final exams on one calendar day
- More than two (2) final exams scheduled in consecutive periods
- Two final exams scheduled for the same exam period.
Given

\[ \text{do } i_1 = L_1, U_1 \]
\[ \ldots \]
\[ \text{do } i_n = L_n, U_n \]

\[ S_1 \quad A(f_1(i_1, \ldots, i_n), \ldots, f_m(i_1, \ldots, i_n)) = \ldots \]
\[ S_2 \quad \ldots = A(g_1(i_1, \ldots, i_n), \ldots, g_m(i_1, \ldots, i_n)) \]

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.

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

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

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

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

<table>
<thead>
<tr>
<th>( \forall D_j )</th>
<th>distance vector</th>
<th>direction vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>( d_1, \ldots, d_{i-1} &gt; 0 )</td>
<td>( d_1, \ldots, d_{i-1} = &quot;&lt;&quot; )</td>
<td></td>
</tr>
<tr>
<td>( d_1, \ldots, d_i = 0 )</td>
<td>( d_1, \ldots, d_i = &quot;=&quot; )</td>
<td></td>
</tr>
</tbody>
</table>
Project and OpenMP

Two important issues while specifying the parallel execution of a `for` loops:

- **safety** – parallel execution has to preserve all dependences
- **profitability** – benefits of parallel execution have to compensate for the overhead penalty
Project and OpenMP

safety

Sample code:

```c
#pragma omp parallel for private(i, hash)
  for (j = 0; j < num_hf; j++) {
    for (i = 0; i < wl_size; i++) {
      hash = hf[j] (get_word(wl, i));
      hash %= bv_size;
      bv[hash] = 1;
    }
  }
```

This specifies:

- outermost (j-loop) is parallel
- each thread will get its own copy of variables `i` and `hash`, eliminating loop carried anti and output dependences.
Project and OpenMP

Sample code:

```c
#define CHUNK_SIZE 2
int chunk = CHUNK_SIZE
#pragma omp parallel for \
    schedule (dynamic, chunk) \
    private(i, hash)
   for (j = 0; j < num_hf; j++) {
      for (i = 0; i < wl_size; i++) {
         hash = hf[j] (get_word(wl, i));
         hash %= bv_size;
         bv[hash] = 1; }
   }
```

This specifies:

- outermost (j-loop) is parallel, with CHUNK_SIZE iterations scheduled as a group; default chunk size=1
- three basic scheduling strategies: **static, dynamic, or guided**

There are many more options of specifying how to execute `for` loops in parallel (see the online OpenMP tutorial)
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

$\Rightarrow$ 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: \[ X(a*i + c1) = \ldots \] \ \ // write
   R2: \[ \ldots X(a*i + c2) \ldots \] \ // 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

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

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:

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

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

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. \( \text{for } i = \text{LB}, \text{UB}, 1 \)
   \[
   \begin{align*}
   \text{R1: } & \quad X(i) = \ldots \quad \text{\textbackslash \textbackslash write} \\
   \text{R2: } & \quad \ldots X(i-2) \ldots \quad \text{\textbackslash \textbackslash read} \\
   \text{endfor}
   \end{align*}
   \]

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

2. \( \text{for } i = \text{LB}, \text{UB}, 1 \)
   \[
   \begin{align*}
   \text{R1: } & \quad X(2*i) = \ldots \quad \text{\textbackslash \textbackslash write} \\
   \text{R2: } & \quad \ldots X(2*i-1) \ldots \quad \text{\textbackslash \textbackslash read} \\
   \text{endfor}
   \end{align*}
   \]

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

Assume R1 executes before R2.

Classification of dependences:

- R1 is write, R2 is read \( \Rightarrow \text{true dependence} \)
- 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:

\[
\text{for } i = \text{LB, UB, 1} \\
\text{R1: } X(c_1) = \ldots \quad \text{\textbackslash write} \\
\text{R2: } \ldots X(c_2) \ldots \quad \text{\textbackslash read} \\
\text{endfor}
\]

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 \textbf{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:

\[ \Delta d \in \{0, \ldots UB - LB\} \text{ for true} \]
\[ \Delta d \in \{1, \ldots UB - LB\} \text{ for anti and output} \]
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.
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)
Loop Interchange

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

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

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

Loop interchange is safe \textit{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

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

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

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

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

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 (Fission)

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

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

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

Loop distribution is safe \textit{iff}

- statements involved in a cycle of true deps (\textit{recurrence}) remain in the same loop, and

- if \exists a dependence between two statements placed in different loops, it must be forward

\[\Rightarrow \text{ Benefits} \]

- necessary for vectorization
- may enable partial/full parallelization
- may enable other loop transformations
- may reduce register/cache pressure
Next Lecture

Things to do:

• What do you want to talk about?
• Open floor for questions.