Class Information

- Project 3 posted.
- Homework 8 will be posted after this lecture.
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 \textit{dependence} between statement $S_1$ and $S_2$, denoted $S_1 \delta S_2$, indicates that $S_1$, the \textit{source}, must be executed before $S_2$, the \textit{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: 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 ≠ 0.

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

Zheng Zhang
Review: 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:

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

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

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

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

1. for $i = \text{LB}, \text{UB}, 1$
   
   R1: $X(i) = \ldots$ \hspace{1cm} \text{% write}
   R2: $\ldots = X(i - 2) \ldots$ \hspace{1cm} \text{% read}

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

2. for $i = \text{LB}, \text{UB}, 1$
   
   R1: $X(2 \times i) = \ldots$ \hspace{1cm} \text{% write}
   R2: $\ldots = X(2 \times i - 1) \ldots$ \hspace{1cm} \text{% read}

   $a=2, c_1=0, c_2=-1 \Rightarrow \Delta d = \frac{1}{2}$ (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
Distance Vectors

Distance Vector: number of iterations between accesses to the same memory location

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

\begin{tabular}{|c|c|c|}
\hline
\textbf{Distance Vector} & \textbf{Distance} & \textbf{Direction} \\
\hline
\textit{S}_1(wt A) \delta \textit{S}_1(rd A) & (0,1) & (=,<) \\
\textit{S}_2(wt A) \delta \textit{S}_2(rd A) & (1,1) & (<,<) \\
\textit{S}_3(wt B) \delta \textit{S}_3(rd B) & (1,-1) & (<,>) \\
\hline
\end{tabular}
Which Loops are Parallel

- 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 there is no dependence vector $D$ carried at level $i$ for some pair of memory references.

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

- Allows expression of parallelism at different levels: task and loop level
- Parallelization is done through pragmas.
- Look at the OpenMP documentation on our class web site.
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
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.
#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

There are many more options of specifying how to execute
for loops in parallel (see the online OpenMP tutorial)
Vectorization V.S. Parallelization

Revisit this example:

Exercise: which loop is parallelizable or vectorizable?

do I = 1, 100
do J = 1, 100
A(I,J) = A(I,J) + 1
enddo
enddo

parallelization

doall I = 1, 100
doall J = 1, 100
A(I,J) = A(I,J) + 1
enddo
implicit barrier sync.
enddo

vectorization

Left Loop: \( A(1:100:1,1:100:1) = A(1:100:1,1:100:1) + 1 \)
Right Loop: \( A(1:99,1:100) = A(2:100,1:100) + 1 \)
Case Study I: A Vectorizing Source-to-Source 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;
}
```
Case Study I: A Vectorizing Source-to-Source Compiler

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)
5. no function calls
SKETCH OF BASIC ALGORITHM

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
   2.1 if SCC has more than one statement in it, distribute loop around it and generate sequential code
   2.2 if SCC is a single statement and has no true dependence cycle, distribute loop around it and generate vector code; otherwise, generate sequential code
EXAMPLE

for (i=2; i<99; i++) {
    S1:   a[i] = b[i-1] + c[i-1] + 3;
    S2:   b[i] = (c[i] + b[i+1]) / 2;
    S3:   c[i] = a[i] + 1;
    S4:   d[i] = b[i] + c[i+1];
}

Zheng Zhang
EXAMPLE

S2: \( b[2:99] = \frac{(c[2:99] + b[3:100])}{2}; \)
    for (i=2; i<99; i++) {
        S1: \( a[i] = b[i-1] + c[i-1] + 3; \)
        S3: \( c[i] = a[i] + 1; \)
    }
Loop Transformation

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

```plaintext
do I = 1, N
  do J = 1, N
    S_1  A(I,J) = A(I,J-1)
    S_2  B(I,J) = B(I-1,J-1)
  enddo
enddo

⇒ loop interchange ⇒

do J = 1, N
  do I = 1, N
    S_1  A(I,J) = A(I,J-1)
    S_2  B(I,J) = B(I-1,J-1)
  enddo
enddo
```

Loop interchange is safe *iff*

- it does not change the sign of the lexicographical direction vector, i.e., (1,-1) → (-1,1)

⇒ Benefits
  - may increase parallel loop granularity
  - reordering iterations may improve reuse
Loop Fusion

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

⇒ loop fusion ⇒

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

Loop fusion is safe \textit{iff}

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

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

⇒ Benefits
  - reduces loop overhead
  - improves reuse between loop nests
  - increases granularity of parallel loop
Loop distribution is safe \textit{iff}

\begin{itemize}
  \item statements involved in a cycle of true deps (\textit{recurrence}) remain in the same loop, and
  \item if $\exists$ a dependence between two statements placed in different loops, it must be forward
\end{itemize}

$\Rightarrow$ Benefits: necessary for vectorization, may enable partial/full parallelization, may enable other loop transformations, may reduce register/cache pressure
Next time:

- Matrix Matrix Multiplication, Locality Optimization and GPU Computing