INFORMATION and REMINDERS

• Project 1 has been graded
• Homework 7 due tonight
• Project 2
  – Deadline extension? Monday, April 24, 11:59pm
  – Use **plt-r5rs** - racket version v6.5 (command line interpreter); do not include **#lang racket** from DrRacket.

• **FINAL EXAM**
  1. **Monday, May 8, 4:00-7:00pm**, College Ave. Campus
  2. **CONFLICTS?** Need to know as soon as possible; there are fixed rules to resolve conflicts; **Deadline: May 1**
Dependence Analysis for Array 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

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

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

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

\[1 \leq I \leq 5\]
\[I \leq J \leq 6\]

- 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 \( I' = (i'_1, \ldots, i'_n) \),

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

\begin{align*}
    & \text{Distance Vector} = \text{number of iterations between accesses to the same location} \\
    & \text{Direction Vector} = \text{direction in iteration space (}=, <, >) \\
    & \text{distance vector} \quad \text{direction vector}
\end{align*}

\begin{align*}
    S_1 & \quad \delta S_1 \\
    S_2 & \quad \delta S_2 \\
    S_3 & \quad \delta S_3
\end{align*}
Which Loops are Parallel?

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

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

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

\[
\begin{align*}
\bullet & \text{ a dependence } D = (d_1, \ldots, d_k) \text{ is carried at level } i, \text{ if } d_i \text{ is the first nonzero element of the distance/direction vector}
\end{align*}
\]

\[
\begin{align*}
\bullet & \text{ a loop } l_i \text{ is parallel, if } \nexists \text{ a dependence } D_j \text{ carried at level } i
\end{align*}
\]

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

profitability

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)
Next Lecture

Things to do:

• OpenMP tutorial on our website

Remaining topics

• Dependence analysis
• Loop level source-to-source optimizations
• Automatic vectorization