REMINDERS

• Sample solutions of HW#6 and HW#7 have been posted. Sample solution for HW#8 will be posted later today. No late submission for HW#8.

• Final exam: Thursday, December 17, 4:00pm - 7:00pm, BRR 1095; accumulative, closed books, closed notes. For students with cs211 or 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.

YOU MUST take this make-up exam. This is the only make-up offered.

• Still issues with project 1 grade or midterm grade? Come and talk to me after class. Deadline to handle these issues: Thursday, December 10.

• No office hours after December 10.

• Exam review office hour: Monday, December 14,
10:00am to 11:30am, CoRE B (305)

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

```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.
Sample code:

```c
#define CHUNK_SIZE 2
int chunk = CHUNK_SIZE
#pragma omp parallel for \n    schedule (dynamic, chunk) \n    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)
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.
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

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

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

Loop interchange is safe iff

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

⇒ Benefits

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

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

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

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

Loop fusion is safe 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 iff

- statements involved in a cycle of true deps \((\text{recurrence})\) 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
That’s it!

Hope you enjoyed the course!