Lecture 12—Loop-carried Dependencies; Speculation; Parallelization Patterns

ECE 459: Programming for Performance

January 30, 2015

Last Time

Memory-carried dependencies:

		Second Access	
		Read	**Write**
First Access	**Read**	No Dependency Read After Read (RAR)	Anti-dependency Write After Read (WAR)
	Write	True Dependency Read After Write (RAW)	Output Dependency Write After Write (WAW)

We also saw how to break WAR and WAW dependencies.

Plus, loop dependencies, and Mandelbrot example.

Live Coding Demo: Parallelizing Mandelbrot

Refactor the code; create array for output.

Add a struct to pass offset, stride to thread.

Create & join threads.

Part I

Breaking Dependencies with Speculation

Breaking Dependencies

Speculation: architects use it to predict branch targets.

- Need not wait for the branch to be evaluated.

We'll use speculation at a coarser-grained level: speculatively parallelize source code.

Two ways: speculative execution and value speculation.

Speculative Execution: Example

Consider the following code:

```
void doWork(int x, int y) {
    int value = longCalculation(x, y);
    if (value > threshold) {
      return value + secondLongCalculation(x, y);
    }
    else {
      return value;
    }
}
```

Will we need to run `secondLongCalculation`?

Speculative Execution: Example

Consider the following code:

```
void doWork(int x, int y) {
    int value = longCalculation(x, y);
    if (value > threshold) {
      return value + secondLongCalculation(x, y);
    }
    else {
      return value;
    }
}
```

Will we need to run `secondLongCalculation`?

- OK, so: could we execute `longCalculation` and `secondLongCalculation` in parallel if we didn't have the conditional?

Speculative Execution: Assume No Conditional

Yes, we could parallelize them. Consider this code:

```
void doWork(int x, int y) {
  thread_t t1, t2;
  point p(x,y);
  int v1, v2;
  thread_create(&t1, NULL, &longCalculation, &p);
  thread_create(&t2, NULL, &secondLongCalculation, &p);
  thread_join(t1, &v1);
  thread_join(t2, &v2);
  if (v1 > threshold) {
    return v1 + v2;
  } else {
    return v1;
  }
}
```

We do both the calculations in parallel and return the same result as before.

- What are we assuming about `longCalculation` and `secondLongCalculation`?

Estimating Impact of Speculative Execution

T_1: time to run `longCalculatuion`.
T_2: time to run `secondLongCalculation`.
p: probability that `secondLongCalculation` executes.

In the normal case we have:

$$T_{\text{normal}} = T_1 + pT_2.$$

S: synchronization overhead.
Our speculative code takes:

$$T_{\text{speculative}} = \max(T_1, T_2) + S.$$

Exercise. When is speculative code faster? Slower? How could you improve it?