ECE459: Programming for Performance Winter 2015

Lecture 12 — January 30, 2015

Patrick Lam *version 2*

In class, I live-coded a parallelization of the Mandelbrot code. I had to first refactor the code, creating an array to hold the output. Then I added a struct to pass the offset and stride to the thread. Finally, I invoked pthread to create and join threads.

Breaking Dependencies with Speculation

Recall that computer architects often use speculation to predict branch targets: the direction of the branch depends on the condition codes when executing the branch code. To get around having to wait, the processor speculatively executes one of the branch targets, and cleans up if it has to.

We can also use speculation at a coarser-grained level and speculatively parallelize code. We discuss two ways of doing so: one which we'll call speculative execution, the other value speculation.

Speculative Execution for Threads.

The idea here is to start up a thread to compute a result that you may or may not need. 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;
  }
}
```

Without more information, you don't know whether you'll have to execute `secondLongCalculation` or not; it depends on the return value of `longCalculation`.

Fortunately, the arguments to `secondLongCalculation` do not depend on `longCalculation`, so we can call it at any point. Here's one way to speculatively thread the work:

```
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 now execute both of the calculations in parallel and return the same result as before.

Intuitively: when is this code faster? When is it slower? How could you improve the use of threads?

We can model the above code by estimating the probability p that the second calculation needs to run, the time T_1 that it takes to run `longCalculation`, the time T_2 that it takes to run `secondLongCalculation`, and synchronization overhead S. Then the original code takes time

$$T = T_1 + pT_2,$$

while the speculative code takes time

$$T_s = \max(T_1, T_2) + S.$$

Exercise. Symbolically compute when it's profitable to do the speculation as shown above. There are two cases: $T_1 > T_2$ and $T_1 < T_2$. (You can ignore $T_1 = T_2$.)