ECE459: Programming for Performance Winter 2011

Lecture 14 — February 17, 2011

Patrick Lam *version 1*

This isn't a real lecture, because you are busy writing a midterm now, but someone asked to see an example of the use of the OpenMP barrier mechanism, so here you go.

The main source for these notes is "OpenMP 3.0: What's New?"[1], by Alejandro Duran.

First of all, recall that when an OpenMP program hits a *parallel* construct, it creates a team of threads. Implicitly, each thread runs an implicitly-defined task.

"Tied" tasks. Implicit tasks are tied, so that only the thread that starts the implicit task will run it. Untied tasks, once suspended, can be resumed by any thread.

Taskwait. As one might expect, this directive applies to tasks. This pragma tells the currently-executing task to wait until all of its direct child tasks have finished. (It does not wait for grandchildren). Here's an example, from the Solaris Studio OpenMP API User's Guide, Chapter 5[2].

```
#include <stdio.h>
#include <omp.h>
int fib(int n)
{
  int i, j;
  if (n<2)
    return n;
  else
    {
      #pragma omp task shared(i) firstprivate(n)
      i=fib(n-1);

      #pragma omp task shared(j) firstprivate(n)
      j=fib(n-2);

      #pragma omp taskwait
      return i+j;
    }
}

int main()
{
  int n = 10;
```

[1] `http://cobweb.ecn.purdue.edu/ParaMount/iwomp2008/documents/omp30.pdf`, accessed February 16, 2011.

[2] `http://download.oracle.com/docs/cd/E19205-01/821-1381/gkegm/index.html`, accessed March 15, 2011.

```
  omp_set_dynamic(0);
  omp_set_num_threads(4);

  #pragma omp parallel shared(n)
  {
    #pragma omp single
    printf ("fib(%d) = %d\n", n, fib(n));
  }
}
```

Note also the use of the task directive with the shared and firstprivate clauses. The `taskwait` directive tells OpenMP to wait for the two tasks to finish before it continues by reading the values of `i` and `j`.

Barriers. Barriers are like taskwait, but for threads in a team. OpenMP defines implicit barriers at certain places: "all tasks created by any thread of the current team are guaranteed to be completed at the end of the parallel region." You can also write an explicit barrier. Let's extend the example from L13 (untested, but should work):

```
void calc (double *array1, double *array2, int length) {
  #pragma omp parallel for nowait
  for (int i = 0; i < length1; i++) {
    array1[i] += array2[i];
  }
  #pragma omp parallel for nowait
  for (int j = 0; i < length2; i++) {
    array3[j] -= array4[j];
  }
  #pragma omp barrier
  #pragma omp parallel for
  for (int i = 0; i < min(length1, length2); i++) {
    array[i] = array1[i] * array3[i];
  }
}
```

In this case, we want to run both loops in parallel, but we don't wait to wait for the first loop to finish before running the second one. So we run them both in parallel and explicitly state `nowait`; otherwise, OpenMP would put an implicit barrier after each loop. Next, we want to make sure that both loops are done before starting the third loop, so we add an explicit `barrier`.