OpenMP Tasks

Computational Science II (CAAM 520)

Christopher Thiele

Rice University, Spring 2020

Motivation

So far all of our OpenMP code was loop based.

How can we implement more general multi-threaded algorithms?

Tasks

OpenMP tasks allow us to offload any "chunk" of work to a thread.

```
#pragma omp parallel
{
  #pragma omp single
  {
    #pragma omp task
    foo();

    #pragma omp task
    bar();
  }
}
```

Tasks

In this example `foo()` and `bar()` will be executed as tasks.

The OpenMP runtime environment schedules the tasks:

- Tasks can be executed by any thread.
- Tasks can be executed in any order.

→ Why do we need a `single` directive inside the parallel region?

Data environment for tasks

A task views variables as

- `firstprivate`, if the variable was `private` to the thread that created the task.
- `shared`, if the variable was `shared` in the thread that created the task.

Data environment for tasks

```
int i;

#pragma omp parallel
{
  int j;

  #pragma omp task
  {
    // Task has a "shared" view of i.
    // Task has a "firstprivate" copy of j.
  }
}
```

Data environment for tasks

The data environment can become complicated.

To be safe, use defaults!

```
int i;
#pragma omp parallel
{
  int j;
  #pragma omp task shared(i) firstprivate(j) \
                   default(none)
  {
    // The default(none) clause requires that
    // any variable is declared as shared or
    // firstprivate *explicitly*!
  }
}
```

Tasks and barriers

An OpenMP barrier enforces the completion of ***all*** incomplete tasks that were created in the current parallel environment.

```
#pragma omp parallel
{
  #pragma omp single
  for (int i = 0; i < 4; i++) {
    #pragma omp task
    do_work(i);
  }
  #pragma omp barrier

  // All tasks are complete at this point.
}
```

The `taskwait` directive

The `taskwait` directive enforces the completion of all ***child*** tasks.

It does not synchronize threads in any other way, i.e., it is weaker than a barrier.

Tasks and dependencies

OpenMP allows us to express dependencies between tasks:

```
#pragma omp parallel
{
  #pragma omp single
  {
    #pragma omp task depend(out:x)
    x = foo();
    // First task must finish before
    // this task can run.
    #pragma omp task depend(in:x)
    bar(x);
  }
}
```

Tasks and dependencies

Dependencies can be expressed by adding `depend(type:list)` clauses to the `task` directive, where

- `type` can be `in`, `out`, `inout`, among other options, and
- `list` is a comma-separated list of variables.

→ Instead of individual variables, we can also specify ranges within arrays in the format `my_array[start:length]`.