CSC346/646 Parallel Computation Spring 2015

OpenMP and Lightweight Threads

Pragmas

- Parallel constructs are implemented by compiler directives (pragmas).
- By default the compiler ignores the directives and produces a sequential executable.
- A compiler option (-fopenmp) enables the pragmas and produces a parallel executable.

Concepts

- Concurrent threads are created at the beginning of a parallel section.
- Threads terminate at the end of a parallel section.
- Threads are all running the same (translated source) code in the parallel section
- OS maps threads onto processor cores, hopefully one thread per core.
- Threads run in a shared address space, but ....
- Each thread has a private run-time stack.
  - Discussion: implications for automatic variables
  - Discussion: implications for function calls
  - Discussion: private vs shared pointers

Variables

- All variables are either shared or private.
- A shared variable refers to the same memory location in the sequential section of the program and in all parallel sections.
- A private variable occupies a different memory location for each thread in the parallel section, and also a different memory location from the same variable in the sequential portion of the program.
- Variables not declared either shared or private are shared by default.

Control-Flow and Synchronization

- Library functions provide a unique ID number for each thread, numbered $0, 1, 2, ...n-1$, and the number of threads.
- An environment variable (`OMP_NUM_THREADS`) controls the number of threads at run-time.
- Different thread behavior can be based on ID number, or
- OpenMP pragmas can be used to:
  - Distribute an iteration space over the threads
  - Barrier synchronization
  - Critical section