CS 33 – Discussion 1A
Computer System Organization

Week 7

Questions before we start?

Logistics

- Lab 3 – Attack lab - Due tonight 11:59pm
- Lab 4 – parallel lab – OpenMP multithreading
  - Likely released this weekend – Due end of week 10
  - HW 4 and HW5 will be released/due before then
- Please submit suggestions on CCLE TA-site on how to improve OH/Disc.
  - Please fill out LA survey on CCLE if you haven't already

Agenda

10am-11am PST:

- Overview of multi-threading/OpenMP
- Intel slides until we run out of time

11am-11:50am PST:

- LA worksheet

Multi-threading basics

- **Ways to exploit parallelism :**
  - Domain Decomposition - Dividing *data* to be processed amongst processors/threads
  - Task Decomposition - Dividing *tasks* to be completed amongst processors/threads
  - Pipelining - Multiple instructions executed in parallel by dividing task into stages and having different processors/threads execute different stages simultaneously

Multi-threading basics

- **Fork-Join programming model:**
  - A programming model that facilitates incremental parallelism by forking into threads at parallel portions of the program and joining the threads at sequential portions of the program.
  - Every thread has its own cache where it can store private variable values (thread-local storage), but they also have access to shared code and data segments

Multi-threading basics

Race Condition:

- When two or more threads access a shared resource at the same time, potentially causing undesired behavior. The results are non-deterministic.

Mutual Exclusion:

- Restricting access to a shared resource to one thread at a time based on a locking mechanism. Creates atomic accesses.

Lock:

- A synchronization mechanism/construct that facilitates exclusive access to a shared resource by a single process/thread at a time
    - Mutex Locks
    - Spin Locks
    - Semaphores - flags

Multi-threading basics

Critical Section:

- A piece of code that threads should execute in a mutually-exclusive fashion, or in other words: sequentially (don't parallelize)

Deadlock:

- Multiple threads want each other's locks but won't release their own locks until obtaining the other's
- i.e. Thread A has lock a and Thread B has lock b
  - Thread A wants lock b and Thread B wants lock a -> *Deadlock*

Conditions for deadlock:

- Threads perform mutually exclusive access to a shared resource
- Threads with locks hold onto them while waiting for desired locks
- No Pre-emption – cant take locks away from threads
- Cycles in resource allocation graph imply resource waiting

OpenMP basics

OpenMP:

- A library that helps add parallelism to your programs using compiler directives (pragmas)

API summary:

- Functions:
  - omp_get_num_procs()
  - omp_set_num_threads()
  - omp_get_num_threads()
  - omp_init_lock(omp_lock_t* lock)
  - omp_set_lock(omp_lock_t* lock)
  - omp_unset_lock(omp_lock_t* lock)

OpenMP basics

API summary:

- Pragmas:
  - #pragma omp parallel num_threads(k) - spawns k threads
  - #pragma omp for - divides loop iterations for the following loop between spawned threads
  - #pragma omp parallel for - spawns threads and then divides loop iterations amongst the spawned threads
  - #pragma omp single - only one of the spawned threads should execute the following code block

OpenMP basics

API summary:

- Pragmas:
  - #pragma omp nowait - threads that have completed execution of the code block need not wait for other threads to complete before proceeds
  - #pragma omp critical - Demarcates a critical section
  - #pragma omp atomic - Executes the following section atomically (guarantees serialization only for certain operations)
  - #pragma omp (parallel) sections & #pragma omp section - execute demarcated blocks of code (sections) in parallel

OpenMP basics

API summary:

- Clauses:
  - private([list of variables]) - makes private copies of the specified variables for each spawned thread
  - firstprivate([list of variables]) - makes private copies that inherits the values of the shared variables
  - lastprivate([list of variables]) - assigns the private values to the shared values after the parallel region completes
  - reduction([OP]:[variable]) - makes private copies of variable for each thread, performs OP computation on each private copy, and then combines all private copies (representing local results) into global copy (representing global result)

Questions before we start worksheet?

Works Cited

- Professor Reinman's slides (CCLE)
- LAs – Sidharth Ramanan, Julia Baylon, Sana Shrikant et al.

- **Credit for compilation: Attiano Purpura-Pontoniere**