Thread Pool with Multiple Queues Solutions

Thread Pool with Multiple Queues

- In the thread pool in the last lecture, all the threads shared the same queue of tasks
- Are there any disadvantages to this?
  - To prevent a data race, all operations on the queue must be locked
  - When one thread takes a task off the queue, it locks the queue
  - Other threads are blocked until this operation is complete
  - If there are many small tasks, this can affect performance
- How does giving each thread its own queue of tasks help here?
  - When a thread gets a task, it cannot be blocked by another thread which is getting a task
  - A thread never has to wait to get its the next task

Work Sharing

- Does this "work sharing" approach give better performance than a single queue?
  - Yes, if the time lost due to being blocked is significant in comparison to the time taken to perform a task
- Are there any disadvantages to work-sharing?
  - An uneven distribution of tasks can cause sub-optimal performance
  - If a thread has no tasks in its queue, it will be idle
  - Even if other threads have tasks to do

Work Sharing Implementation

- Suggest how to convert the thread pool from the previous lecture to use work-sharing
  - Replace the single queue by a fixed-size vector of queues
  - The vector will have one element for each thread in the pool
  - Provide a scheduling mechanism to distribute tasks among these queues

Work Sharing Implementation

- What is meant by "round-robin" scheduling?
  - Tasks are given to each thread in turn
  - The first task goes on the first thread's queue, the second task on the second queue, and so on
  - After putting a task on the last thread's queue, go back to the first thread's queue