CSE201 Tutorial-8

Multithreading

Parallelize Divide-and-Conquer Mergesort Algorithm

In this tutorial our task is to parallelize the divide-and-conquer (recursive) implementation of Merge Sort algorithm in two ways as mentioned below

1. Parallelize using explicit threading
   - Create as many number of threads as the number of tasks you are generating
2. Parallelize using ForkJoinPool and Java ExecutorService
   - Create tasks and push to the task pool
   - Threads from thread pool will take these tasks out of task queue and will execute it

Once you have both these implementations ready, execute them one by one by increasing the number of threads and observe the execution time. How many threads are you able to generate the first version?

Parallelize Divide-and-Conquer Mergesort Algorithm

- Copy the sequential version of this divide-and-conquer algorithm from following link
    - https://docs.google.com/document/d/1viRx_NwsQinS5yKa8xH_8I8KO0SROe-wnCKVzwYvbFo/edit?usp=sharing
- Parallelize this sequential version in two different ways as mentioned in slide-2

Hint-1: Identifying Parallel Tasks

- Which is the most compute intensive method call?
  - mergesort(…….)

- Which are parallel tasks in this algorithm
  - mergesort for left half of array ?
  - mergesort for right half of array ?
  - merge method ?

Hint-2: Joining Tasks / Threads

- join() needs to be called in both these parallel versions
  - Thread.join()
  - RecursiveAction.join()
- Where you should call join() in both parallel implementation?

Hint-3: We have already seen something similar during lectures

- Explicit threading version
  - See the parallel array sum example in Lecture 18 slide-10
- Java ExecutorService version
  - See the parallel array sum example in Lecture 19 slide-8
- ForkJoinPool version
  - See the parallel Fibonacci example in Lecture 19 slide-17