CS2030S Recitation Problem Set 9

Brian Cheong

ForkJoinPool

ForkJoinPool

- Parallel divide and conquer
- Break up the problem into smaller problems
- Combine the results
- Achieved with `RecursiveTask<T>`

RecursiveTask

- `fork` : Add to the head of the deque (other dudes can pick it up from behind)
- `join` : 2 cases: 1) if done read result 2) call `compute`
- `compute` : execute task (which may or may not fork depending on size)
- When thread is idle,
  - check if OWN deque empty if not take from head
  - steal work from the tail of other threads deque

Order of fork and join

- After forking, join in reverse order
- Because if not will need to do some pops and push to get to the subtask we want
    - Less efficient if done this way

Question 1

- Trace thru the events
  - What tasks get added to the deque?
  - Which worker executes which task?
  - Which worker steals which task?
- Brian will now show the code and run a few times

Question 1

- Output differs from run to run
- all task except count = 4 will be sent to deque
- whichever worker is free will execute the task (seemingly random)
- When a worker waits on a join, it can go steal other work from other worker

Question 2

```
import java.util.concurrent.RecursiveTask;

class Fibonacci extends RecursiveTask<Integer> {
    private final int x;
    Fibonacci(int x) {
        this.x = x;
    }
    @Override
    protected Integer compute() {
        if (this.x <= 1) {
            return 1;
        }
        Fibonacci f1 = new Fibonacci(this.x - 1);
        Fibonacci f2 = new Fibonacci(this.x - 2);

        // decide the affects of the ordering of forking
        :
    }
}
```

Question 2a

Code

```
f1.fork();
int a = f2.compute();
int b = f1.join();
return a + b;
```

Analysis

- f1 is forked for other workers to complete
- f2 is completed by the current thread
- f1.join is like waiting for f1 to be done in case it's not

Question 2b

Code

```
f1.fork();
int a = f1.join();
int b = f2.compute();
return a + b;
```

Analysis

- f1 is forked for other workers to complete
- f1.join waits for the entire f1 to finish
- f2.compute is done by the current thread
- no parallelism

Question 2c

Code

```
int a = f1.compute();
int b = f2.compute();
return a + b;
```

Analysis

- f1.compute is done on the current thread
- f2.compute is done sequentially after f1 by the current thread
- no parallelism

Question 2d

Code

```
f1.fork();
f2.fork();
int a = f2.join();
int b = f1.join();
return a + b;
```

Analysis

- f1.fork allows other workers to work on it
- f2.fork allows other workers to work on it as well but f2 is on the head
- f2.join gets the result from f2
- f1.join gets the result from f1
- allows f1 and f2 to run in parallel

Question 2e

Code

```
f1.fork();
f2.fork();
int a = f1.join();
int b = f2.join();
return a + b;
```

Analysis

- f1.fork allows other workers to work on it
- f2.fork allows other workers to work on it as well but f2 is on the head
- f1.join gets the result from f2
  - need to find f1 on the deque
- f2.join gets the result from f2
- allows f1 and f2 to run in parallel but less efficient

That's all folks

It was an honour and pleasure to teach all of you

❤️

That's all folks

It was an honour and pleasure to teach all of you ❤️

all the best for exams

I'll miss you guys (maybe)