Revision Lecture on Concurrent Programming

Lecture 14 of TDA384/DIT391

Principles of Concurrent Programming

Nir Piterman and Gerardo Schneider

Chalmers University of Technology | University of Gothenburg

UNIVERSITY OF GOTHENBURG

CHALMERS UNIVERSITY OF TECHNOLOGY

DISCLAIMER!

This is NOT a sample of how the exam will be

It's just a revision of some of the topics we have seen in the course

Outline

- Mentimeter on concurrency topics
- One question from an old exam

Mentimeter

Review of topics in concurrency

https://www.mentimeter.com/s/0569b7837a2a99b0219a1bed74a94559/ef3edec180dd/edit

Old Exam Question

We have seen the following parallel implementation of merge sort

(lecture 09,
slides 21, 22, and 25):

```java
public class PMergeSort extends RecursiveAction {
  private Integer[] data;
  private int low, high;

  @override
  protected void compute() {
    if (high - low <= 1) {
      sort(data,low,high); // sort sequentially small chunks of 1024
      return;              // or less
    }
    int mid = low + (high - low)/2; // mid point
    // left and right halves
    PMergeSort left = new PMergeSort(data,low,mid);
    PMergeSort right = new PMergeSort(data,mid,high);
    left.fork();  // fork thread working on left
    right.fork(); // fork thread working on the right
    left.join();  // wait for sorted left half
    right.join(); // wait for sorted right half
    merge(mid);   // merge halves
  }
}
```

The following appears somewhere in the `main`:

```
RecursiveAction sorter = new PMergeSort(numbers,0,numbers.length);
ForkJoinPool.commonPool().invoke(sorter);
```

Based on the dependency graph (or otherwise) for a run of `invoke(sorter)` when the array numbers has 8 elements, answer the following.

(Part a). How many threads participate in the computation? *(4p)*

(Part b). What is the maximum number of tasks that can be executed in parallel in this implementation on the same data (excluding parent tasks waiting for a child task to finish)? *(4p)*

You apply the second optimization in slide 25. That is, you change line 16 to `right.compute();` and comment out line 18.

(Part c). How many threads participate now in the computation? *(4p)*

(Part a). How many threads participate in the computation? *(4p)*

Answer: there are $8 + 4 + 2 + 1 = 15$ *nodes.*

1: original thread
2: first call
4: second call
8: third call

(Part b). What is the maximum number of tasks that can be executed in parallel in this implementation on the same data (excluding parent tasks waiting for a child task to finish)? *(4p)*

Answer: 8 calls – the width of the dependency graph.

You apply the second optimization in slide 25. That is, you change line 16 to `right.compute();` and comment out line 18.

(Part c). How many threads participate now in the computation? *(4p)*

```java
1  public class PMergeSort extends RecursiveAction {
2    private Integer[] data;
3    private int low, high;
4
5    @override
6    protected void compute() {
7      if (high - low <= 1) {
8        sort(data,low,high); // sort sequentially small chunks of 1024
9        return;              // or less
10     }
11     int mid = low + (high - low)/2; // mid point
12     // left and right halves
13     PMergeSort left = new PMergeSort(data,low,mid);
14     PMergeSort right = new PMergeSort(data,mid,high);
15     left.fork();  // fork thread working on left
16     right.compute();
17     left.join();  // wait for sorted left half
18     right.join(); // wait for sorted right half
19     merge(mid);   // merge halves
20   }
21 }
```

You apply the second optimization in slide 25. That is, you change line 16 to `right.compute();` and comment out line 18.

(Part c). How many threads participate now in the computation? *(4p)*

Answer: There are 8 nodes. There is one thread that does the work that 4 threads did previously, one thread that does the work that 3 threads did previously, and two threads that do the work that 4 threads (2 for each) did previously. Overall the number of saved threads is $3 + 2 + 1 + 1 = 7$.

[Not considering the father thread, there 14 threads, and by replacing right.fork with right.compute we eliminate half of those]

(Part d). What is the maximum number of tasks that can be executed in parallel? *(3p)*

(Part e).

You now get an array with 9000 elements. Change the program according to the first advice in slide 25 so that the number of threads that participate in the computation does not change to all the previous answers. *(4p)*

[To set a threshold (different from 1 on when to start sorting)]

(Part d). What is the maximum number of tasks that can be executed in parallel? *(3p)*

Answer: This is still 8 – the width of the dependency graph.

(Part e).

You now get an array with 9000 elements. Change the program according to the first advice in slide 25 so that the number of threads that participate in the computation does not change to all the previous answers. *(4p)*

```
public class PMergeSort extends RecursiveAction {
  private Integer[] data;
  private int low, high;

  @override
  protected void compute() {
    if (high - low <= 1) {
      sort(data,low,high); // sort sequentially small chunks of 1024
      return;              // or less
    }
    int mid = low + (high - low)/2; // mid point
    // left and right halves
    PMergeSort left = new PMergeSort(data,low,mid);
    PMergeSort right = new PMergeSort(data,mid,high);
    left.fork();  // fork thread working on left
    right.fork(); // fork thread working on the right
    left.join();  // wait for sorted left half
    right.join(); // wait for sorted right half
    merge(mid);   // merge halves
  }
}
```

New threshold!

(Part e).

You now get an array with 9000 elements. Change the program according to the first advice in slide 25 so that the number of threads that participate in the computation does not change to all the previous answers. *(4p)*

Answer: Line 7 should be changed to, for example:

```
if (high - low <= 1200) {
```

Why 1200?

We need to find a number that makes the recursion stop after 3 calls (in order to keep the same number of threads as before): (((9000/2)/2)/2) = 1125
(so any number between 1125 and 2249 would make it)

Exam: Monday March 13 at 14:00 (Johanneberg)

Remember: Material permitted during the exam (*hjälpmedel*):

- Two textbooks
- four sheets of A4 paper with notes
- English dictionary
- NOTHING MORE!

- NOTE: **You cannot bring photocopies of the books!**

Please answer
the
Course Survey