Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Parallel Programming
Assignment 2: Introduction to Multi-threading
Spring Semester 2020

Assigned on: **26.02.2020**

Due by: **(Wednesday Exercise) 02.03.2020**
(Friday Exercise) 04.03.2020

Overview

This week's assignment is about simple multi-threaded programs in Java.

- Download the ZIP file named `assignment2.zip` on the course website.
- Import the project in Eclipse: Click on *File* in the top-menu, then select *Import*. In the dialog, select *Existing Projects into Workspace* under the *General* directory, then click on Next. In the new dialog, select the radiobox in front of *Select archive file* to import a ZIP file. Then, click Browse on the right side of the text-box to select the ZIP file you just downloaded from the website (`assignment2.zip`). After that, you should see assignment2 as a project under *Projects*. Click Finish.
- If you have done everything correctly, you should now have a project named assignment2 in your *Package Explorer*.

Task 1 – Loop Parallelization

For this week's programming exercise you should learn how to improve the performance of executing a loop. For this purpose we will use threads to parallelize its execution.

Description: Our goal in this exercise will be to parallelize the execution of the following loop defined in `computePrimeFactors` method:

```
for (int i = 0; i < values.length; i++) {
  factors[i] = numPrimeFactors(values[i]);
}
```

which computes the number of prime factors for each element in an given array. For example, for number `12` the number of prime factors is `numPrimeFactors(12) = 3` since `12 = 2*2*3`. The implementation of `numPrimeFactors` is already provided for you in the assignment template and should not be changed.

Notice: Java libraries not included in the assignment project are not allowed. Do not rename any of the provided methods in the assignment (you can create additional classes and methods).

Tasks

A) To start with, print to the console `"Hello Thread!"` from a new thread. How do you check that the statement was indeed printed from a thread that is different to the main tread of your application? Furthermore, ensure that you program (i.e., the execution of main thread) finishes only after the thread execution finishes.

B) Run the method `computePrimeFactors` in a single thread other than the main thread. Measure the execution time of sequential execution (on the main thread) and execution using a single thread. Is there any noticeable difference?

C) Design and run an experiment that would measure the overhead of creating and executing a thread. That is, the time it takes to create a thread object with empty runnable, calling the `start` method and waiting for the thread to finish execution.

D) Before you parallelize the loop in task E), design how the work should be split between the threads by implementing method `PartitionData`. Each thread should process roughly equal amount of elements. Briefly describe your solution and discuss alternative ways to split the work.

E) Parallelize the loop execution in `computePrimeFactors` using a configurable amount of threads.

F) Think of how would a plot that shows the execution speed-up of your implementation, for $n = 1, 2, 4, 8, 16, 32, 64, 128$ threads and the input array size of $100, 1000, 10000, 100000$ look like.

G) Measure the execution time of your parallel implementation for $n = 1, 2, 4, 8, 16, 32, 64, 128$ threads and the input array size of $100, 1000, 10000, 100000$. Discuss the differences in the two plots from task F) and G).

To understand your results, it may be a good idea to find out how many cores your system has available. In Java, you can query the Runtime class to find this information:

```
int cores = Runtime.getRuntime().availableProcessors();
```

Creating Threads in Java

Java offers a couple of possibilities to create new threads. Below we briefly cover the most common options. Regardless of the variant used to create the Thread object, we always call the `start()` method on thread objects to actually run the associated threads.

The most compact option is to create the code that is run by a thread anonymously and inline:

```
// create thread
Thread t = new Thread(){
  public void run(){
    // code to execute in the thread
  }
};
// start thread
t.start();
```

The second – and most flexible option – is to separate the task you want to accomplish and the execution of said task. This can come in handy if you have a computation that relies on a lot of shared state but is at the same time fairly complicated. In that scenario you want to the class that implements your task to implement `Runnable`. Using this method you can share a single task object between multiple threads:

```
public class MyOperation implements Runnable {
  private final int initialValue;

  public MyOperation(int initialValue) {
    this.initialValue = initialValue;
  }

  public void run() {
    // code to execute when run
  }

  // can define more methods and fields here
}
```

Now we can create threads using a `MyOperation` object:

```
int initVal = 5;
MyOperation op = new MyOperation(initVal);
Thread t = new Thread(op);
t.start();
```

It is also possible to create a custom class that inherits from Thread:

```
public class MyThread extends Thread {
  // thread local data
  private final int initialValue;

  public MyThread(int initialValue) {
    this.initialValue = initialValue;
  }

  public void run(){
    // code to execute in thread
  }

  // can define more methods and fields here
}
```

We can then create a new thread using a new instance of our custom class:

```
// create thread
int initValue = 5;
Thread t = new MyThread(initValue);
// start thread
t.start();
```

Measuring Time

In Java you can measure time in nano-second increments using the snippet listed below.

```
long startTime = System.nanoTime();

// the code you want to measure time for goes here

long endTime = System.nanoTime();
long elapsedNs = endTime - startTime;
double elapsedMs = elapsedNs / 1.0e6;
```

Submission

In order to receive feedback for your exercises, you need to submit your code to the Git repository. You will find detailed instructions on how to install and set-up Eclipse for use with Git in Exercise 1.

Once you have completed the skeleton, commit it to Git by following the steps described below. For the questions that require written answers, please write them on paper and bring them to the next exercise session where the solutions will be discussed.

- **Check-in your project for the first time**
  - Right click your created project called **assignment2**.
  - In the menu go to **Team**, then click **Share Project**.
  - You should see a dialog Configure Git Repository. Here, next to the Repository input field click on **Create...**
  - Select a root git directory or your projects that you have created in Execise 1. Note for all your assignments you should use the same directory.
  - click **Finish**.
- **Commit changes in your project**
  - Now that your project is connected to your git repository, you need to make sure that every time you change your code or your report, at the end you commit your changes and send (push) them to the git server.
  - Right click your project called **assignment2**.
  - In the menu go to **Team**, then click **Commit..**.
  - In the Comment field, enter a comment that summarizes your changes.
  - In the Files list, select all the files that you changed and want them to be committed. This typically includes all the Java files but not necessarily all the files (e.g., you dont have to commit setting files of our eclipse installation).
  - Then, click on **Commit** to store the changes locally or **Commit and Push** to also upload them to the server. Note that in order to submit your solution you need to **both** commit and push your changes to the server.
- **Push changes to the git server**
  - Right click your project called **assignment2**.
  - In the menu go to **Team**, then click **Push Branch 'master'**. Note if this is not your fist push you can also use **Push to Upstream** to speed up the process.
  - A new dialog appears, now fill in for the URL field:
    `https://gitlab.inf.ethz.ch/COURSE-PPROG20/<nethz-username>.git`
  - Click **Next**
  - Keep the default values and click **Next**
  - An authentication dialog should appear. Fill in your nethz username and password and click **OK**.
  - Click **Finish** to confirm your changes. Note that eclipse might ask for authentication again.
- **Browse your repository online**
  - you can access and browse the files in your repository online on GitLab at:
    `https://gitlab.inf.ethz.ch/COURSE-PPROG20/<nethz-username>`