ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Parallel Programming
Assignment 6: Task Parallelism
Spring Semester 2020

Assigned on: **25.03.2020**

Due by: **(Wednesday Exercise) 30.03.2020**
(Friday Exercise) 1.04.2020

Overview

This week's assignment is about Task Parallelism. Task parallelism emphasizes the distributed (parallelized) nature of the processing (i.e., threads), as opposed to the data (data parallelism). We will use the fork/join framework to implement task parallelism. The fork/join framework helps to take advantage of multiple processors. It is designed for work that can be broken into smaller pieces recursively. The goal is to use all the available processing power to enhance the performance of your application.

Getting Prepared

- Download the ZIP file named `assignment6.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 (`assignment6.zip`). After that, you should see assignment6 as a project under *Projects*. Click Finish.
- If you have done everything correctly, you should now have a project named assignment6 in your *Package Explorer*.

1 Sorting

In this exercise you are asked to implement merge-sort algorithm using task parallelism. The merge-sort algorithm sorts an array by partitioning it in smaller arrays. Once the size of the arrays becomes 1 or 2, they are trivially sorted. Sorted sub-arrays are combined by *merging* them.

The simplest task in merge-sort is the trivial sort of small arrays. This is the following:

```
if (array.length == 1) {

  // An array of size 1 is implicitly sorted
  return array;

} else if (array.length == 2) {

  // Re-arrange elements if they are not in proper order
  if (array[0] > array[1]) {
    temp = array[0];
    array[0] = array[1];
    array[1] = temp;
  }
  return array;

}
```

Merging two sorted arrays into a bigger sorted array is also a simple procedure. Consider the following two sorted arrays:

```
array1: [5, 11, 12, 18, 20]
array2: [2, 4, 7, 11, 16, 23, 28]
```

The resulting merged array is computed by comparing the head of the two arrays. The smallest one is removed and it is appended at the end of the output. In this case, the merged output is:

```
result: [2, 4, 5, 7, 11, 11, 12, 16, 18, 20, 23, 28]
```

The code to merge two ordered arrays is already provided to you in `ethz.ch.pp.util.ArrayUtils`. Futher, we also provide an sequential merge-sort in `ethz.ch.pp.mergeSort.MergeSortSingle`.

Your task is to implement the task-parallel version of merge-sort.

Task A: The code you downloaded and imported contains relevant parts in `MergeSortMulti` that you are expected to implement. For merge-sort with task parallelism, the following is one of the possible strategies to divide the work:

```
sort(input) {

  if (input.length <= 2) {

    // Execute the simple task locally
    return simpleSort(input);

  } else {

    // Split the input in two parts by forking to two tasks

    fork(firstPart);
    fork(secondPart);

    waitForResults();

    // Join results
    return merge(firstPartResults, secondPartResults);

  }

}
```

Implement the strategy outlined above using the `ForkJoinPool` and `RecursiveAction` Java APIs.

Task B: Verify that the multi-threaded merge-sort computes the correct results by running the provided set of test-cases.

2 Longest Sequence

In this exercise our goal is to find the longest sequence of the same consecutive number in an input sequence of numbers. For example, we show the longest sequences for a given input below:

$$[1, 9, 4, 3, 3, 8, 7, 7, 7, 0] \xrightarrow{\text{longest sequence}} [7, 7, 7]^{end:8}_{start:6}$$

Where the longest sequence is formed by three consecutive numbers 7 that start at index 6 and end at index 8. For all non-empty inputs the longest sequence always contains atleast one element. In case of multiple sequences having the same length we always return the one with smaller starting index. We illustrate both of these cases with following two examples:

$$[0, 1, 2, 3, 4, 5, 6, 7, 8, 9] \xrightarrow{\text{longest sequence}} [0]^{end:0}_{start:0}$$
$$[1, 1, 0, 0] \xrightarrow{\text{longest sequence}} [1, 1]^{end:1}_{start:0}$$

We provide a sequential version that returns the longest sequence of the same consecutive number in `LongestCommonSequence` class. You may assume that the input array has always atleast one element.

Task A: Implement a task parallel version that computes longest sequence using the fork/join framework in `LongestCommonSequenceMulti` class. Start with a cutoff set to value 2. Note that in this task we cannot simply split the input array into two partitions whose results can be computed independently and combined afterwards. For example, if we would split input $[1, 3, 3, 2]$ into two parts $[1, 3]$ and $[3, 2]$ we would miss the longest sequence $[3, 3]$ as it is on the boundary. However, for many inputs (such as $[3, 3, 1, 2]$) we can safely split them and compute the results independently. Make sure you implementation handles this case and successfully parallelizes the computation without producing incorrect results. Use the provided set of test cases to verify your implementation.

Task B: Improve the performance of the implementation from Task 1 by choosing more appropriate cutoff value. Compare the performance to the sequential version. Note that the computation performed in the base case (e.g., comparing that two array values are the same) is very simple and fast. To make the task more compute-intensive, use more expensive comparison (e.g., `Math.exp(i) == Math.exp(j)` instead of `i == j`).

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 **assignment6**.
  - 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 **assignment6**.
  - 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 **assignment6**.
  - 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>`*