Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Parallel Programming
Assignment 6: Task Parallelism II
Spring Semester 2017

Assigned on: **28.03.2017** Due by: **03.04.2017**

Overview

This week's assignment provides additional exercises that demonstrate advanced usage of Task Parallelism.

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 Task Parallelism with Memoization

In the first task we will investigate the parallelization of the well known task of generating Fibonacci numbers using the fork/join framework. Mathematically, the *n*-th Fibonacci number is defined using following recursive formula (assuming that $n \geq 0$):

$$F_n = \begin{cases} n & \text{if } n \leq 1 \\ F_n = F_{n-1} + F_{n-2} & \text{otherwise} \end{cases}$$

Using this equation we can easily compute the first Fibonnaci numbers, e.g., $F(0) = 1$, $F(1) = 1$, $F(2) = 2$, $F(3) = 3$, $F(4) = 5$, $F(5) = 8$, etc. We provide the implementation of the above formula in `FibonacciSeq` class.

Task 1: Create a multi-threaded version in skeleton class `FibonacciMulti` using the fork/join framework (e.g., using the `ForkJoinPool`). As can be seen from the formula, the computation of *n*-th Fibonacci number naturally decomposes into two smaller subtasks. Experiment with different cutoff values and check that the runtime decreases compared to the sequential version.

Task 2: Although we implemented task-parallel computation of Fibonacci numbers in Task 1, you noticed that the computed is inherently wasteful. That is, the same subproblems are being computed multiple times. For example, the computation of $F(4)$ is as follows:

$$\begin{aligned} F(4) &= F(3) + F(2) \\ &= (F(2) + F(1)) + (F(1) + F(0)) \\ &= ((F(1) + F(0)) + F(1)) + (F(1) + F(0)) \\ &= 3 * F(1) + 2 * F(0) \end{aligned}$$

where values $F(1)$ and $F(0)$ are computed three and two times respectively.

To address this issue you implement an optimization technique called memoization that stores the results of computed Fibonacci numbers. Then, when the same number is computed again we simply return the cached value and avoid repeating the expensive computation. Your task is to implement this technique (e.g., use appropriate data structure to store and retrieve computed values) in skeleton class `FibonacciMultiCache`. Note, that your implementation needs to be properly synchronized as there will be multiple threads accessing the cache at the same time.

Task 3: Apply the memoization technique also to the sequential implementation in `FibonacciSeqCache`. Compare the runtime results of various versions you have implemented. Discuss what speed-up (if any) you achieved and what might be the reasons behind your measurements.

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 1: 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 2: 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 for us to grade your exercises and give you feedback, you need to submit your code to the Subversion repository. You will find detailed instructions on how to install and set-up Eclipse for use with Subversion in Exercise 1.

Once you have completed the skeleton, commit it to SVN in a directory named `assignment6` by following the steps described below. The questions that require written answers should all be recorded in a single file named `report.pdf` and placed in the base directory of your project (i.e., in folder `assignment6`).

- **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**.
  - In the dialog that now appears, select **SVN** as a repository type, then click **Next**.
  - In case you have submitted Exercise 1, choose **Use existing repository location** and select the pre-defined URL in the dialog that should look like this
    https://svn.inf.ethz.ch/svn/vechev/pprog17/students/NETHZ_USERNAME
    Click Finish. Otherwise follow the steps in Exercise 1 to set-up a repository location.
- **Commit changes in your project**
  - Now that your project is connected with the SVN server, you need to make sure that every time you change your code or your report, at the end you submit it to the SVN server as well.
  - 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.
  - Then, click on **Ok**.