SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Established in collaboration with MIT

Computer System Engineering

Week 3: Lab 2 (40 marks)

Objective: Run Computing Tasks using Multi-thread

The objective of this lab is to understand the concept of threads, and to use the **multi-thread** paradigm to speed up certain computing processes. Another objective is to observe the performance gain from using this paradigm.

Submission deadline: 23:59 Feb 22, 2017.

You may choose to do this lab in either Java or C. Section 1 provides the details of the tasks in Java. Section 2 provides the details of the tasks in C.

Section 1. JAVA version (lab 2)

Task 1: Find Mean of an Array Using Multi-thread in Java (15 marks)

The goal of task 1 is to find the arithmetic mean of the elements in an array. Given an array with length M, first partition it into N parts (each part is a sub-array with length M/N). The mean value for each of the N sub-arrays is found by using the concept of multi-thread. Each sub-array, is assigned to a thread which is tasked to find the mean of that sub-array. After obtaining the N mean values of the sub-arrays in a separate array, the global mean is determined by calculating the mean of the values in this array.

You will need to implement an extended class of *java.lang.Thread* for finding the mean value. Define your function to find mean value, and call it in the function *run()*.

The suggested steps for the algorithm are as follows:

(1) Read data from "*input.txt*" (length = 524288), and store it in an array; note that the size of the array is a power of 2. This is to facilitate partitioning. The file "*input.txt*" contains a list of integers separated by space. You can use Java utilities, e.g., "Scanner", to read data from the file and store it into an array.

(2) Partition the array into N parts, and run N threads to find corresponding mean values of each part (do your experiment with thread number N=1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048);

(3) Print out intermediate values, i.e., print out the N mean values;

(4) Find and printout the global mean value from the N mean values (should be ~5001.93);

(5) Record the execution time taken for different number of threads (see the following table), and plot the graph (number of threads versus execution time);

Number of threads	1	2	4	8	16	32	64	128	256	512	1024	2048
Execution time (ms)												

Task 2: Find Median of an Array Using Multi-thread in Java (25 marks)

The second task is to find the median of an array. The procedure is similar to Task 1, but computing median involves performing sort and merge operations on the array before the median can be determined. As in the case of the mean value, you are advised to implement an extended class of *java.lang.Thread* for this task.

Given an array of size M, first partition it into N parts (each part is a sub-array with length M/N). Then sort each of the N sub-arrays using any sorting algorithm you prefer. After you obtain the N sorted sub-arrays, merge them into one sorted array. This array should be sorted in ascending order. You are allowed to use either in-place sorting (i.e. you do not use any additional memory), or out-of-place sorting. For the merge algorithm, you are allowed to use either a single thread or multi-thread algorithm. Once merged, you need to compute the median of the array (you are allowed to use either single thread or multi-thread algorithm to compute the median).

The suggested steps for the algorithm are as follows:

(1) Read data from "*input.txt*" (length = 524288), and store it in an array;

(2) Partition the array into N sub-arrays, and run N threads for storing each sub-array (do your experiment with thread number N=1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048);

(3) Merge the N sorted sub-arrays into one sorted array (both single and multi-thread are permitted);

(4) Print out the sorted array;

(5) Compute median of the array (should be 5006.0);

(6) Record execution time of sorting, merging and computing median (use the following table) and plot the execution time (execution time versus number of threads). Discuss how the execution time changes based on the number of threads.

Number of threads	1	2	4	8	16	32	64	128	256	512	1024	2048
Execution time (ms)												

The starting codes and input file for Task 1 and Task 2 can be found on eDimension:

MeanThread_StartingCode.java
MedianThread_StartingCode.java
input.txt

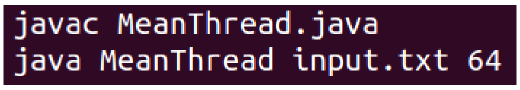
Suggested functions for Java:

Function	Used for	Class which function belongs to
.start()	Starting a thread by calling its run() method	Thread
.join()	Waiting for a thread to terminate	Thread
run()	An entry point for the thread	Thread
.subList	Returning a specific portion of a list	ArrayList<Integer>
.currentTimeMillis()	Getting current time in MS	System

Tips: how to compile and run your Java code?

***Compile**: javac   your_code.java*
***Run**:   java your_code file_name number_of_threads*

Example:



```
javac MeanThread.java
java MeanThread input.txt 64
```

This example shows that we are using MeanThread.java to find mean value of the data in file "input.txt" using 20 threads.

Submission instruction: after completing Task 1 and Task 2, submit the two java files (modified from "*MeanThread_StartingCode.java*" and "*MedianThread_StartingCode.java*"), and a doc file (containing mean and median values obtained, the execution time table, plot of the execution time and your discussion on number of threads verus execution time) to eDimension **before 23:59 Feb 22, 2017!**

Section 2. C version (lab 2)

Task 1: Find Mean Value of an Array Using Multi-thread in C (15 marks)

The first task is to find the mean value of all the items in an array.

Given an array with length M, first partition it into N parts (each part is a sub-array with length M/N). Find the mean value for each of the N sub-arrays using multi-thread. For each sub-array, put it in a thread, start the thread to find the mean value, and output the mean value. After that, get the N mean values corresponding to the N threads, and put them in another array. Finally find the mean value of this array.

It is recommended that you use two functions (*get_temporal_mean(), get__global_mean())* for finding mean value. *get_temporal _mean*() is to get mean value for each sub-array and store them in *temp_result[]*. Then use *get_global_mean()* to find the final mean value from *temp_result[]* array. You need to define your function to find mean value, and assign that function to the task for each thread. Use the pthread library in C to work with threads.

The suggested steps for the algorithm are as follows:

(1) Read data from "*input.txt*" (length = 524288), and store it in an array (array length should be 524288); note that the size of the array is a power of 2. This is to facilitate partitioning. The file "*input.txt*" contains a list of integers separated by space. You can use C builtin functions, e.g., "fopen()" and "fscanf()", to read data from the file and store it into an array.

(2) Partition the array into N parts, and run N threads to find corresponding mean values of each part (do your experiment with thread number N=1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048) and store each mean value to temp_result[];

(3) Print out intermediate values, i.e., print out the N mean values;

(4) Find the global mean value from *temp_result[]* (should be ~5001.93);

(5) Record the execution time taken for different number of threads (see the following table), and plot the graph (number of threads versus execution time).

Number of threads	1	2	4	8	16	32	64	128	256	512	1024	2048
Execution time (ms)												

Task 2: Find Median of an Array using Multi-thread in C (25 marks)

The second task is finding median of an array with multi-thread, sorting an array using sort and merge with multi-thread. The procedure is similar to Task 1, but computing median involves sorting and merging of the array first.

Given an array with length M, first partition it into N parts (each part is a sub-array with length M/N). Then sort each of the N sub-arrays using any sorting algorithm. After that, get the N sorted sub-arrays, and merge them into one sorted array. The array should be sorted ascendingly. Both in-place and out of place sorting strategies are allowed. For the merge algorithm, both single thread and multi-thread algorithms are allowed.

Given an array with length M, first partition it into N parts (each part is a sub-array with length M/N). Then sort each of the N sub-arrays using any sorting algorithm you prefer. After that, get the N sorted sub-arrays, and merge them into one sorted array. The array should be sorted in ascending order. Both in-place and out of place sorting strategies are allowed. For merge algorithm, both single thread and multi-thread algorithms are allowed. Once merged, we need to compute median of the array (both single thread and multi-thread are allowed to compute median). Use the pthread library in C to work with threads.

The suggested steps for the algorithm are as follows:

(1) Read data from "*input.txt*" (length = 524288), and store it in an array (array length should be 524288);

(2) Partition the array into N parts, and run N threads for for sorting (similarly, do your experiment with thread number N=1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048);

(3) Merge the N sorted sub-arrays into one sorted array (single or multi-thread is OK);

(4) Print out the sorted array;

(5) Compute median of the array (should be 5006.0);

(6) Record execution time of sorting, merging and computing median (use the following table) and plot the execution time (execution time verus number of threads). Discuss how the execution time changes based on the number of threads.

Number of threads	1	2	4	8	16	32	64	128	256	512	1024	2048
Execution time (ms)												

The starting codes and input file for Task 1 and Task 2 can be found on eDimension:

MeanThread_StartingCode.c
MedianThread_StartingCode.c
input.txt

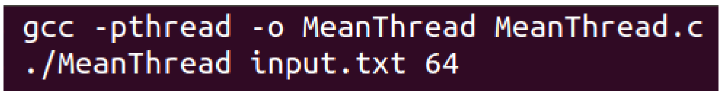
Suggested functions for C:

Class/Function	Used for	Needed headers
pthread_t	*Define thread*	*#include <pthread.h>*
pthread_create	*Create thread*	*#include <pthread.h>*
pthread_join	*Wait thread finish*	*#include <pthread.h>*
pthread_exit()	*Exit thread*	*#include <pthread.h>*
clock_t	*Define time*	*#include <time.h>*
clock()	*Get time*	*#include <time.h>*

Tips: How to compile and execute in c:

Compile: *gcc -pthread -o executable_file_name your_code.c*
Run: *./executable_file_name input.txt number_of_threads*

Example:



```
gcc -pthread -o MeanThread MeanThread.c
./MeanThread input.txt 64
```

Submission instruction: after completing Task 1 and Task 2, submit the two C files (modified from "*MeanThread_StartingCode.c*" and "*MedianThread_StartingCode.c*"), and a doc file (containing mean and median values obtained, the execution time table, plot of the execution time and your discussion on number of threads verus execution time) to eDimension **before 23:59 Feb 22, 2017!**