SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Established in collaboration with MIT

Computer System Engineering

Week 2: Lab 2

Objective: Run Computing Tasks using Multi-thread

To understand multi-thread, and use multi-thread to speed up certain computing process, see the performance gain from using multi-thread.

Submission: before 23:59 at 9th February 2016

Q1. Find Max Value of an Array Using Multi-thread in Java

The first question is about finding the max value of an array. It is a simple question and it is ok if you want to use Java built-in function to do it (e.g. *Collection.max()*). The main purpose is to learn how to use multi-thread.

Given an array with length M, we first partition it into N parts (each part is a sub-array with length M/N). Then we find max value for each of the N sub-array using multi-thread. For each sub-array, we put it in a thread, start the thread to find the max value, and output the max value. After we get the N max values correspond to N threads, we put them in an array, and finally we find the max value of this array.

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

Your code should have the following function:

(1) Read data from "*input_1.txt*" (length = 100), store the data in an array;

(2) Partition the array into N parts, and run N threads for finding max value (N=2, 5, 10, 20, 50);

(3) Print out the N max values;

(4) Find the max value from the N max values;

(5) Print out the max value (should be 99).

Suggested Function:

Function	Class which function belongs to
.subList	*ArrayList<Integer>*

.start()	*Thread*
.join()	*Thread*

Q2. Sort an Array using Multi-thread in Java

The second question is about sorting an array using sort and merge with multi-thread. The procedure is similar with Q1, except that we change the task within each thread. You need to separate the target array into N parts, and sort them in N threads (using any sorting algorithm). Then merge the N sorted sub-arrays into final array.

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

Similar with Q1, we implement an extended class of *java.lang.Thread* for sorting.

Your code should have the following function:

(1) Read data from "*input_1.txt*" (length = 100), store the data in an array;

(2) Partition the array into N parts, and run N threads for sorting (N=2, 5, 10, 20, 50);

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

(4) Print out the sorted array;

(5) Record the execution time for the sorting part of your program;

(6) Read data from "*input_2.txt*" (length is 500000), change the number of threads (set N=2, 5, 10, 20, 50, 100, 200, 500, 1000, 2000), record the execution time, and explain how execution time changes based on number of threads (close the print out function when you record execution time).

You can record the execution time in the following table:

Number of threads	1	2	5	10	20	50	100	200	500	1000	2000
Execution time (ms)											

After you finish this table, write down your analysis about the relationship between number of threads and execution time.

Suggested Function:

Function	Class which function belongs to
.currentTimeMillis()	*System*

The starting codes and data files for Q1 and Q2 can be found in eDimension:

MultiThread - Q1 - starting code.java

MergeSortThreaded - Q2 - starting code.java

input_1.txt

input_2.txt

Tips: **how to compile and run your Java code?**

Compile: *javac your_code.java*

Run: *java your_code file_name threadNumber*

Example:

This example shows we are using MergeSortThreaded2 to sort file "input_2.txt" in 20 threads.

After you finish Q1 & Q2, submit the two java files (modified from "*MultiThread - Q1 - starting code.java*" and "*MergeSortThreaded - Q2 - starting code.java*") and a doc file (contains the execution time table and your analysis) to eDimension **before 23:59 at 9th February 2016**.

Lab2: C version

Q1. Find Max Value of an Array Using Multi-thread in C

The first question is about finding the max value of an array. The main purpose is to learn how to use multi-thread.

Given an array with length M, we first partition it into N parts (each part is a sub-array with length M/N, care about the situation of M%N != 0). Then we find max value for each of the N sub-array using multi-thread. For each sub-array, we put it in a thread, start the thread to find the max value, and output the max value. After we get the N max values correspond to N threads, we put them in an array, and finally we find the max value of this array.

Here we need to implement two functions (*get_max_from_array(),get_max_from_result())* for finding max value. For *get_max_from_array*() is to get the maximum value for sub-arrays and store them in *temp_result[]*. Then use *get_max_from_result()* to find the maximum value within *temp_result[]* array, its result should be the maximum of whole array. Define your function to find max value, and bond them with each thread.

Your code should have the following function:

(1) Read data from "*input_1.txt*", store the data in an array (array length should be 100);

(2) Print out the N max values;

(3) Find the max value from the N max values, and store result in *temp_result[];*

(5) Find the max value from *temp_result[];*

(5) Print out the max value (should be 99).

Suggested Function for C:

Class/Function	Used for	Needed headers
pthread_t	*define*	*#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>*

Q2. Sort an Array using Multi-thread in C

The second question is about sorting an array using sort and merge with multi-thread. The procedure is similar with Q1, except that we change the task within each thread. You need to separate the target array into N parts, and sort them in N threads (using any sorting algorithm). Then merge the N sorted sub-arrays into final array.

Given an array with length M, we first partition it into N parts (each part is a sub-array with length M/N, care about the situation of M%N != 0). Then we sort each of the N sub-array using *sorter().* After we get the N sorted sub-arrays, we merge them into one sorted array.

The array should be sorted ascendingly. Both in-place and out of place sorting strategies are allowed. For merge algorithm, both single thread and multi-thread algorithms are allowed.

Your code should have the following function:

(1) Read data from "*input_1.txt*", store the data in an array (array length should be 100);

(2) Partition the array into N parts, and run N threads for *sorter()* (N=2, 5, 10, 20, 50);

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

(4) Print out the sorted array;

(5) Record the execution time for the sorting part of your program;

(6) Read data from "*input_2.txt*" (array length should be 500000), change the number of threads (set N=2, 5, 10, 20, 50, 100, 200, 500, 1000, 2000), record the execution time, and explain how execution time changes based on number of threads (close the print out function when you record execution time).

You can record the execution time in the following table:

Number of threads	1	2	5	10	20	50	100	200	500	1000	2000
Execution time (ms)											

After you finish this table, write down your analysis about the relationship between number of threads and execution time.

Suggested Function for C:

Class/Function	Used for	Needed headers
clock_t	*define*	*#include <time.h>*
clock()	*Get time*	*#include <time.h>*

The starting codes and data files for Q1 and Q2 can be found in eDimension:

Multithread1_starting_code.c

MergeSortThreaded2_starting_code.c

input_1.txt

input_2.txt

Tips: **How to compile and execute:**

Compile: *gcc -pthread MergeSortThreaded2.c -o Merge2*

Run: *./Merge2*

example:

```
/C$gcc -pthread MergeSortThreaded2.c -o Merge2
/C$./Merge2
```

After you finish Q1 & Q2, submit the two C files (modified from "*Multithread1_starting_code.c*" and "*MergeSortThreaded2_starting_code.c*") and a doc file (contains the execution time table and your analysis) to eDimension **before 23:59 at 9th February 2016**.