2021.1 Multicore Computing, Project #1

(Due : April 26th 11:59pm)

Submission Rule

1. Create a directory "proj1". In the directory, create two subdirectories, "problem1" and "problem2".
2. In each of the directory "problem1" and "problem2", Insert (i) JAVA source code, (ii) a document that reports the parallel performance of your code, and (iii) readme.txt. The document that reports the parallel performance should contain (a) in what environment (e.g. CPU type, memory size, OS type ...) the experimentation was performed, (b) **tables and graphs** that show the execution time (unit:milisecond) for the number of entire threads = {1,2,4,6,8,10,12,14,16,32}. (c) The document should also contain **explanation/analysis on the results and why such results are obtained with sufficient details**. (d) The document should also contain **your entire JAVA source code and screen capture image of program execution and output**. In **readme.txt** file, you should briefly explain how to compile and execute the source code you submit. You should use JAVA language.
3. zip the directory "proj1" into "proj1.zip" and submit the zip file into eClass homework board.

※ If possible, please experiment in a PC equipped with a Quad-core CPU.

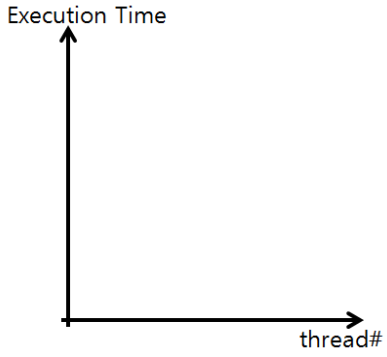
problem 1. Following JAVA program (pc_serial.java) computes the number of 'prime numbers' between 1 and 200000 using a single thread.

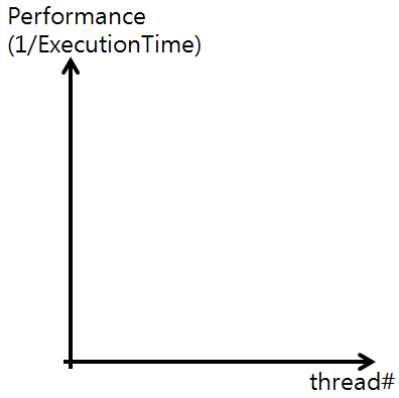
```
class pc_serial {
  private static final int NUM_END = 200000;
  public static void main(String[] args) {
    int counter=0;
    int i;

    long startTime = System.currentTimeMillis();
    for (i=0;i<NUM_END;i++) {
      if (isPrime(i)) counter++;
    }
    long endTime = System.currentTimeMillis();
    long timeDiff = endTime - startTime;
    System.out.println("Execution Time : "+timeDiff+"ms");
    System.out.println("1..."+(NUM_END-1)+" prime# counter=" + counter +"\n");
  }

  private static boolean isPrime(int x) {
    int i;
    if (x<=1) return false;
    for (i=2;i<x;i++) {
      if ((x%i == 0) && (i!=x)) return false;
    }
    return true;
  }
}
```

(i) Implement multithreaded version of **pc_serial.java** using static load balancing and dynamic load balancing, and submit the modified JAVA codes (**"pc_static.java"** and **"pc_dynamic.java"**). Your program should print the (1) execution time of each thread and (2) execution time when using all threads and (3) the number of 'prime numbers'. When writing your JAVA code, NUM_THREAD variable should be used as a constant value in the program (just like pc.java).

FYI, the static load balancing approach performs work division and task assignment while you do programming, which means your program pre-determines which thread tests which numbers. The dynamic load balancing approach lets each thread take a number one by one and test whether the number is a prime number.

(ii) Write a document that reports the parallel performance of your code. The graph that shows the execution time when using 1, 2, 4, 6, 8, 10, 12, 14, 16, 32 threads. There should be at least two graphs, one for static load balancing and the other for dynamic load balancing. Your document also should mention which CPU (dualcore? or quadcore?, hyperthreading on?, clock speed) was used for executing your code.





exec time	1	2	4	...	32
static					
dynamic					

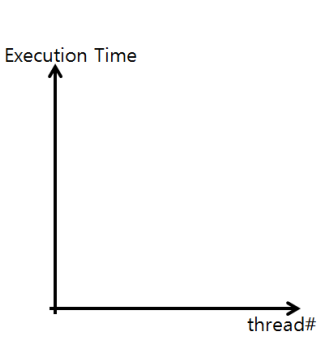
performace (1/exec time)	1	2	4	...	32
static					
dynamic					

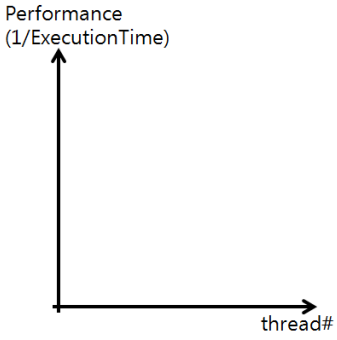
problem 2. (i) Given a JAVA source code for matrix multiplication (the source code **`MatmultD.java`** is available on our class webpage), modify the JAVA code to implement parallel matrix multiplication that uses multi-threads. You should use a static load balancing approach. Your program also should print as output (1) the execution time of each thread, (2) execution time when using all threads, and (3) sum of all elements in the resulting matrix. Use the matrix **`mat500.txt`** (available on our class webpage) as file input (standard input) for the performance evaluation. **`mat500.txt`** contains two matrices that will be used for multiplication.

command line execution example in cygwin terminal> j**`ava MatmultD 6 < mat500.txt`**
In eclipse, set the argument value and file input by using the menu [Run]->[Run Configurations]->{[Arguments], [Common -> Input File].

Here, 6 means the number of threads to use, **`< mat500.txt`** means the file that contains two matrices is given as standard input.

(ii) Write a document that reports the parallel performance of your code. The graph that shows the execution time when using 1, 2, 4, 6, 8, 10, 12, 14, 16, 32 threads. Your document also should mention which CPU (dualcore? or quadcore?, clock speed) was used for executing your code.





	1	2	4	...	32
exec time					

	1	2	4	...	32
performace (1/exec time)					