2017.1 Multicore Computing, Project #1

(Due : April 17th 11:59pm)

Submission Rule

1. Create a directory "proj1". In the directory, create two subdirectories, “problem1” and “problem2”.

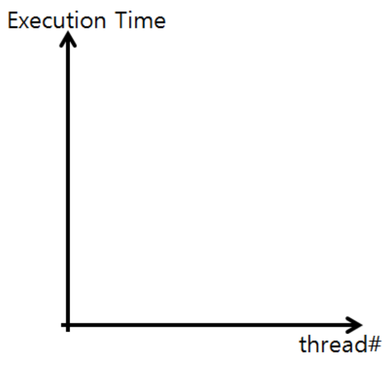
2.1 In each of the directory “problem1” and “problem2”, Insert (i)JAVA source code, (ii)a document that reports the parallel performance of your code, and (iii) readme.txt. The document that reports the parallel performance should contain (a) in what environment (e.g. CPU type, memory size, OS type ...) the experimentation was performed, (b) **tables and graphs** that show the execution time (unit:milisecond) for thread number = {1,2,4,6,8,10,12,14,16}. (c) The document should also contain **explanation on the results and why such results are obtained**. (d) The document should also contain **your entire JAVA source code and screen capture image of program execution**. In **readme.txt** file, you should briefly explain how to compile and execute the source code you submit. You should use JAVA language, but C language using pthread library is also allowed.

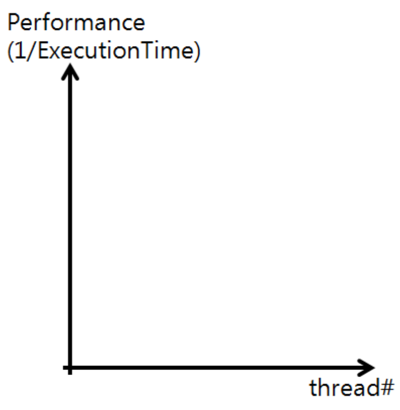
3. zip the directory "proj1" into "proj1.zip" and submit the zip file into eClass homework board.

※ The experimentation (execution of your code and original ex4.java code) should be done **in a quad-core CPU computer. Quad-core CPU PCs are** available in our 4th floor laboratory and etc.

problem 1. In our lab. class, we looked at the JAVA program (ex4.java) that computes the number of ‘prime numbers’ between 1 and 200000. The java code (ex4.java) creates threads for parallel computation using static load balancing approach. However, The naive implementation of ex4.java may not give satisfactory performance because of bad load balancing. The problem is that (i) higher ranges have fewer primes and (ii) larger numbers take longer time to test. Therefore thread workloads may become uneven and hard to predict. For better performance, we consider dynamic load balancing approach where each thread takes a number one by one and test whether the number is a prime number.

(i) Modify ex4.java to adopt dynamic load balancing instead of static load balancing and submit the modified JAVA code (name it "ex4_dynamic.java"). Your dynamic load balancing code should use JAVA synchronization method. Your code also should print the (1) execution time of each thread and (2) execution time for the entire thread computation. When writing your JAVA code, NUM_THREAD variable should be used as a constant value in the program (just like ex4.java).
(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 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?, clock speed) was used for executing your code.





exec time	1	2	4	...	16
static					
dynamic					

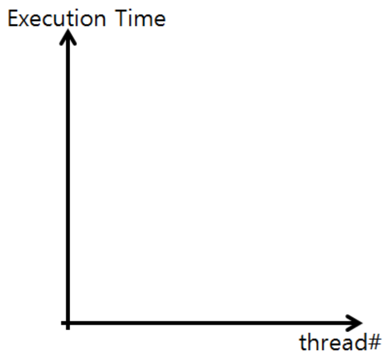
performace (1/exec time)	1	2	4	...	16
static					
dynamic					

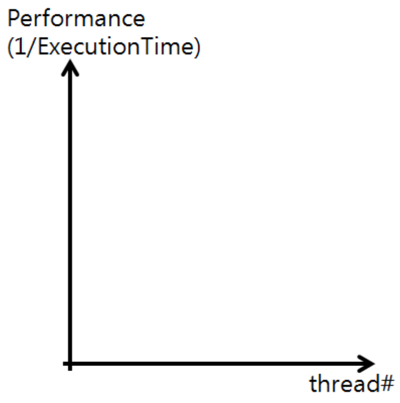
problem 2. (i) Given a JAVA source code for matrix multiplication (the source code MatmultD.java is available on our class webpage), modify the JAVA code to implement parallel matrix multiplication that uses multi-threades. You may use either a static or a dynamic load balancing approach. Your code also should print (1) the execution time of each thread, (2) execution time for the entire thread computation, 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> java 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 threads. Your document also should mention which CPU (dualcore? or quadcore?, clock speed) was used for executing your code.





	1	2	4	...	16
exec time					

	1	2	4	...	16
performace (1/exec time)					