2023.1 Multicore Computing, Project #3

(Due : 11:59pm, May 24)

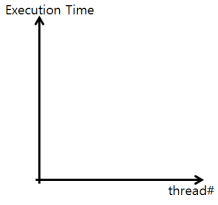
Submission Rule

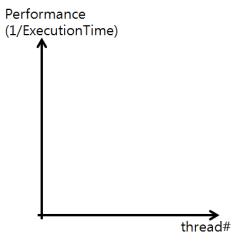
1. Create a directory {studentID#}_proj3 (example: 20203601_proj2). In the directory, create subdirectories 'prob1' and 'prob2'.
2.a For problem 1, write (i)'C with OpenMP' source code **prob1.c**, and (ii)a document that reports the parallel performance of your code into the directory "prob1". Insert the files (i), and (ii) into the subdirectory 'prob1'.
2.b For problem 2, write (i) 'C with OpenMP' source code prob2.c , and (ii) a document that reports the parallel performance of your code. Insert (i) and (ii) into the subdirectory 'prob2'.
2.c For problem 3, insert demo video file (.mp4) into the directory {studentID#}_proj3.
3. zip the directory {studentID#}_proj3 and submit the zip file into eClass homework board.
※ If possible, use quad-core/hexa-core/octa-core CPU (or CPU with more cores) rather than dual-core CPU for your experimentation, which will better show the performance gains of the parallelism.

[Problem 1] In project 1, we looked at the JAVA program that computes the number of 'prime numbers' between 1 and 200000. The parallel implementation of a static approach based on bad work decomposition (i.e. just dividing the entire range of the numbers into *k* consecutive sub-ranges, where *k* is the number of threads) may not give satisfactory performance because (i) higher ranges have fewer primes and (ii) larger numbers are harder (i.e. taking longer time) to test whether they are prime or not. Therefore thread workloads may become uneven and hard to predict. For better performance, we implemented dynamic load balancing approach in project 1 where each thread takes a number one by one and test whether the number is a prime number.

(i) Write 'C with OpenMP' code that computes the number of prime numbers between 1 and 200000. Your program should take two command line arguments: scheduling type number (1 = "static with default chunk size", 2 = "dynamic with default chunk size", 3 = "static with chunk size 10", 4 = "dynamic with chunk size 10"), and number of threads (1, 2, 4, 6, 8, 10, 12, 14, 16) as program input argument. Use **schedule(static) , schedule(dynamic) , schedule(static, 10)** and **schedule(dynamic, 10)**. Your code should print the execution time as well as the number of the prime numbers between 1 and 200000.
command line execution: > **a.out scheduling_type# #_of_thread**
execution example> **a.out 1 8** <---- this means the program use "schedule(static)" using 8 threads.

(ii) Write a document (in PDF file format) 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 four graphs that show the result of static and dynamic scheduling policies. 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 can be obtained**.





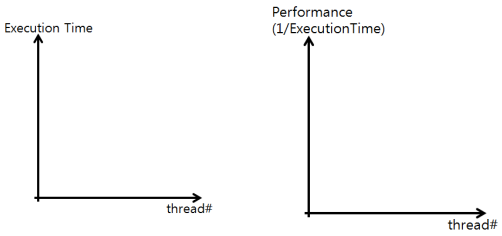
exec time (unit: ms)	chunk size	1	2	4	6	8	10	12	14	16
static	default									
dynamic	default									
static	10									
dynamic	10									

performace (1/exec time)	chunk size	1	2	4	6	8	10	12	14	16
static	default									
dynamic	default									
static	10									
dynamic	10									

[Problem 2] Parallelize prob2.c using OpenMP. Your program should take three command line arguments: scheduling type number (1=static, 2=dynamic, 3=guided), chunk size, and number of threads as program input argument. Your code should print the execution time and the result of PI calculation.
Assume the number of steps **`num_steps`** = 10000000.

command line execution: > **`a.out scheduling_type# chunk_size #_of_thread`**
execution example> **`a.out 2 4 8`** <---- this means dynamic scheduling (chunk size = 4) using 8 threads.

(i) submit the OpenMP source code prob2.c
(ii) Write a document (in PDF file format) that reports the parallel performance of your code. Your report should contain (a) following tables and graphs that shows information in the tables, and (b) brief explanation and interpretation on the results (including why such results can be obtained).



execution time (unit:ms)	chunk size	1	2	4	6	8	10	12	14	16
static	1									
dynamic										
guided										
static	5									
dynamic										
guided										
static	10									
dynamic										
guided										
static	100									
dynamic										
guided										

performace (1/exec time)	chunk size	1	2	4	6	8	10	12	14	16
static	1									
dynamic										
guided										
static	5									
dynamic										
guided										
static	10									
dynamic										
guided										
static	100									
dynamic										
guided										

[Problem 3] Create a demo video file (.mp4 format) that shows compilation and execution of your source files (prob1.c, prob2.c). The size of the demo video file should be less than 50MB.