Operating Systems–2: Spring 2024
Programming Assignment 2 : Thread Affinity
Submission Deadline: 21st February 2024, 9:00 pm

Goal:- To evaluate the efficiency of a parallel matrix squaring algorithm by studying the impact of assigning individual threads to CPU cores.

Details:- In this programming assignment, you will build upon your previous work measuring the performance of calculating the square of matrix A in parallel in C++. Previously, you directly assigned the number of threads to be created, letting the Linux system handle the assignment of cores to the threads using load balancing.

*However, it's known that assigning threads to specific CPU cores, known as **thread affinity**, can enhance cache utilization and reduce cache coherence overhead. Therefore, as part of this assignment, the objective is to assign threads to specific cores and measure performance. For more information, refer to section 5.5.4 from the book Operating System Concepts by Galvin, tenth edition.*

Your task involves testing the performance on both **chunk** and **mixed** designs (explained in assignment 1) with thread affinity. The main program will read the following parameters from an input file: N (number of rows of matrix A), K (number of threads), C (number of logical cores), matrix A, and the number of bounded threads BT.

One can divide the work by creating a thread to compute a row or a collection of rows in the matrix. Then, we execute those threads on different processors in parallel, as performed in the previous assignment.

This assignment involves specifying the number of threads **D** bound to some number of cores and measuring the performance.

Input File:- As mentioned above, the input will consist of four parameters: **N, K, C, the matrix A, and the number of bounded threads BT**. Note that BT <= K. Please name the input file as ***inp.txt.***

The input file is in this format:
N K C BT
A

A sample input file is here:
16 8 4 2
A // Matrix A

Output File:- Your program should output the following in an output file named ***out.txt****: (a) The resulting square matrix* (b) *Total time taken to compute the square matrix*

Report Details:- As part of this assignment, you must prepare a report describing your program's low-level design. You also have to specify the specifications of the system on which you are running the experiments. Further, you have to perform the two experiments described below, measure the performance, and provide the observation you recorded in the tabular format.

Experiment 1: Total Time vs Number of Bounded Threads, BT:
In this experiment, the y-axis of this graph represents the time taken to compute the square matrix. The x-axis will vary with bounded threads BT in terms of a parameter 'b'. The value of BT will vary as b, 2b, 3b, and so forth, up to K, where b = K/C. However, the total number of threads in all the experiments remains the same as K. Thus, the number of bounded threads will be less than K.

Now we explain how the binding should occur. Here, suppose pb threads out of K are bound then: first b threads are bound to core1, the next b threads are bound to core2, etc. The bounded threads are uniformly distributed among their respective cores.

The remaining unbounded threads (K - pb) will be scheduled by the underlying OS scheduler to the total number of cores in the system as per the thread scheduling algorithm.

This graph for this experiment will consist of four curves:

1. Chunk algorithm without threads being bound to cores (as done in the assignment1 experiments)
2. Chunk algorithm with a given b
3. Mixed algorithm without threads being bound to cores (as done in the assignment1 experiments)
4. Mixed algorithm with a given b

Note that since nothing changes for algorithms 1 and 3, the performance of these algorithms should remain the same. The value of b in both of these cases will be 0.

For this experiment, fix N as 1024, and K as 32. The total number of cores C, depends on your laptop. Based on the number of cores C, you can decide b. For instance, if C is 4, then have b as K/C = 8.

Experiment 2: Time vs Number of threads:
In this experiment, you have to compare the performance of the CPU-bound threads with that of normal threads. The x-axis will vary the number of threads from 4 to 64, in the power of 2. The y-axis of the graph represents the average execution time of both types of threads.

Similar to the previous experiment, the number of cores 'C' will depend on your laptop. Here, the number of bounded threads, BT, is equal to K/2. Assign BT threads equally to C/2 cores. The remaining K/2 threads will be assigned to the cores and will be scheduled by the underlying OS scheduler. Measure the average time taken by each type of thread – Bounded threads and Normal threads.

For this experiment, fix N as 1024 and vary K. The total number of cores C depends on your laptop (let's say 8). Based on the number of threads K, you can determine BT. Suppose K is 32, then BT will be 32/2 = 16. Now, you divide 16 threads into 4 cores equally, i.e., each core will be assigned 4 threads. The remaining threads will be scheduled by the OS scheduler on the cores.

Like the above experiment, the graph will consist of six curves, three for chunk methods and three for mixed methods. So, ***for every input on the x-axis, you will record six observations:***

5. Average time execution without threads being bound to cores - Chunk algorithm (as done in the assignment1 experiments)
6. Average time execution of Core-bounded threads - Chunk algorithm
7. Average time execution of Normal threads - Chunk algorithm
8. Average time execution without threads being bound to cores - Mixed algorithm (as done in the assignment1 experiments)
9. Average time execution of Core-bounded threads - Mixed algorithm
10. Average time execution of Normal threads - Mixed algorithm

To handle temporary outliers, ensure that each point in the above plots is averaged over 5 times. Thus, you will have to run your experiment 5 times for each point on the x-axis.

You have to write down the observations you made based on the plots described above and mention any anomalies in the report.

Deliverables:-

You will have to submit the following as part of this assignment:

1. A report describing the following:
(a) Low-level design of your program. You must explain the implementation details correctly.

(b) The graph plots are described above, as well as their analysis. Please name this report file as **Assgn2_Report-<Roll No>.pdf**

2. Prepare a README file that contains the instructions on how to execute your submitted file. The file should be named **Assgn2_ReadMe-<Roll No>.txt**

3. Name the source code file in the following format: **Assgn2_Src-<Roll No>.cpp** (or) **Assgn2_Chunk_Src-<Roll No>.cpp** (or) **Assgn2_Mixed_Src-<Roll No>.cpp**

4. Combine the source code, report, input file, and README as a zip archive file and name it **Assgn2-<Roll No>.zip**. Upload this zip file.
Please see the instructions given above before uploading your file. **Please follow this naming convention.** Your assignment will **NOT be evaluated** if there is any deviation from the instructions posted there, especially concerning the naming convention.

Marking Scheme:

The evaluation scheme for this assignment will be as follows:

S. No	**Section**	**Marks**
1.	Program Design & Report	50
2.	Program Execution	40
3.	Code Indentation and Documentation	10
Total		**100**

Please keep in mind that all the submissions are subjected to plagiarism checks.