CS2518 - P4 - THREADS

In this practical it is the aim to write a program for matrix multiplication. The program should carry out the task to multiply matrix A with B and produce C as result as follows:

$$C = A \times B = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & c_{34} \\ c_{41} & c_{42} & c_{43} & c_{44} \end{bmatrix}$$

where

$$c_{ij} = \sum_{k=1}^{4} a_{ik} b_{kj}$$

The aim is to write a program that can make use of multiple processors (if available!) in the machine executing the program. Thus, the task of matrix multiplication should be broken down such that it can be distributed on multiple processors.

A starting point of the program in c is given below. The program create matrix A and B and populates these with random values. The code to perform the multiplication is missing and you should implement this part.

Each element c_{ij} in the target matrix can be calculated independently and so for the example above 16 independent threads could be used to perform the calculation in parallel. Given that our lab machines do not have 16 cores we can reduce the number of threads by assigning each thread the computation of one matrix row; in this case we need 4 threads.

The matrix data structures are global variables that are visible to all threads. However, as threads will only read from matA and matB and then only write independent rows of the result matrix matC we will be able to complete this task without mutex to protect these shared data structures.

At the point where it states *your code goes here !* write code to create the appropriate number of threads using *pthread_create()* and provide the row number to be calculated as parameter for the thread. Then write a function for the thread to execute the multiplication task.

Use a print statement in each thread to display the CPU ID on which the thread is executing (you can get the CPU id on Linux using *sched_getcpu()*). Increase *MAX* and observe how your threads distribute across the CPU cores.

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sched.h>
int sched_getcpu(void);

// maximum size of matrix
#define MAX 4

int matA[MAX][MAX];
int matB[MAX][MAX];
int matC[MAX][MAX];

int main() {
    // generating random values in matA and matB
    for (int i = 0; i < MAX; i++) {
        for (int j = 0; j < MAX; j++) {
            matA[i][j] = rand() % 10;
            matB[i][j] = rand() % 10;
        }
    }
    // displaying matA
    printf("Matrix A:\n");
    for (int i = 0; i < MAX; i++) {
        for (int j = 0; j < MAX; j++)
                printf("%d ", matA[i][j]);
        printf("\n");
    }
    // displaying matB
    printf("Matrix B:\n");
    for (int i = 0; i < MAX; i++) {
        for (int j = 0; j < MAX; j++)
                printf("%d ", matB[i][j]);
        printf("\n");
    }

    // your code goes here!

    // displaying the result matrix
    printf("Matrix C:\n");
    for (int i = 0; i < MAX; i++) {
        for (int j = 0; j < MAX; j++)
                printf("%d ", matC[i][j]);
        printf("\n");
    }
    return 0;
}
```

CS2518 Continuous Assessment - PART 4

Please submit your completed and commented program.

In addition, answer the following question in detail: *Why does a GPU provide a significant performance gain compared to a multi-core CPU when multiplying large matrices?*

The program and your answer has to be submitted at the end of the semester together with questions from the other practicals (You do not have to complete all tasks/answers within the practical slot).