**Kennesaw State University**

**CS 7172 Parallel and Distributed Computing - Spring 2020**

**Project 2 - OpenMP**

Instructor: Kun Suo

Points Possible: 100

Due date: check on the D2L

The following code implements multiplication of two matrices. The order of the matrix is 2048. Function matrixInit() initializes a double type value for all elements in the matrix. Function matrixMulti() performs the multipy calculation. However, the program executes in the sequential implementation.

<https://github.com/kevinsuo/CS7172/blob/master/Matrix_Multiple_Sample.c>

----------------------------------------------------------------------------------------------------------------

#include <stdio.h>

#include <omp.h>

#include <time.h>

#include <stdlib.h>

#define N 2048

#define FactorIntToDouble 1.1;

double firstMatrix [N] [N] = {**0.0**};

double secondMatrix [N] [N] = {**0.0**};

double matrixMultiResult [N] [N] = {**0.0**};

void matrixMulti()

{

**for**(int row = **0** ; row < N ; row++){

**for**(int col = **0**; col < N ; col++){

double resultValue = **0**;

**for**(int transNumber = **0** ; transNumber < N ; transNumber++) {

resultValue += firstMatrix [row] [transNumber] \* secondMatrix [transNumber] [col] ;

}

matrixMultiResult [row] [col] = resultValue;

}

}

}

void matrixInit()

{

**for**(int row = **0** ; row < N ; row++ ) {

**for**(int col = **0** ; col < N ;col++){

srand(row+col);

firstMatrix [row] [col] = ( rand() % **10** ) \* FactorIntToDouble;

secondMatrix [row] [col] = ( rand() % **10** ) \* FactorIntToDouble;

}

}

}

int main()

{

matrixInit();

clock\_t t1 = clock();

matrixMulti();

clock\_t t2 = clock();

printf("time: %ld", t2-t1);

**return** **0**;

}

----------------------------------------------------------------------------------------------------------------

**Task 1 (50 points):**

Write a parallel program using OpenMP based on this sequential solution.

To compile the program with OpenMP, use:

$ gcc program.c -o program.o -fopenmp

Please write a one-page report (with number and figures), which compares the execution time of sequential solution and parallel solution under different matrix orders (value of N).

**Task 2 (50 points):**

In order to further improve the performance, the matrix can be divided into blocks, and a part of the matrix can be calculated at one time. Under such the implementation, the CPU can move a part of the matrix data into the cache, which can improve the cache hit rate and the program performance.

Please write a block-optimized matrix multiplication program and use OpenMP to parallel its execution. Compare the program execution time with that in Task 1 and write another report with data and figures.

**Submitting Assignment**

Submit your assignment zip file through D2L using the appropriate assignment link. For task 1 and 2, please submit the ***source code*** , ***screenshot of output*** and ***report***.