**TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI**

**VIỆN TOÁN ỨNG DỤNG VÀ TIN HỌC**

**Logo, company name

Description automatically generated**

**BÁO CÁO MÔN**

**TÍNH TOÁN SONG SONG**

**Giảng viên hướng dẫn:**

TS. Đoàn Duy Trung

**Thành viên nhóm:**

**Họ và Tên MSSV**

Nguyễn Hữu Thuật 20185410

Trang Hải Long 20185382

Trần Xuân Hiếu 20185354

Lại Tiến Long 20185376

**CHƯƠNG TRÌNH**

#include <stdio.h>

#include <stdlib.h>

#include <omp.h>

#include <chrono>

#include <iostream>

constexpr auto NUM\_THREADS = 8;

using namespace std;

using namespace std::chrono;

// !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

void DummyDataInitialization(double\* pMatrix, int Row, int Col) {

int i, j; // Loop variables

srand(unsigned(clock()));

for (i = 0; i < Row; i++)

{

for (j = 0; j < Col; j++)

pMatrix[i \* Col + j] = 3.0 + rand() % 2000000;

}

}

// !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

void ProcessInitialization(double\*& pMatrix, double\*& pResult, double\*& pSerialResult, int& Row, int& Col) {

// row and col of initial matrix and vector definition

do {

printf("\nEnter row of the initial objects: ");

scanf\_s("%d", &Row);

printf("\nChosen objects row= %d\n", Row);

if (Row <= 0)printf("\nRow of objects must be greater than 0!\n");

} while (Row <= 0);

do {

printf("\nEnter Colum of the initial objects: ");

scanf\_s("%d", &Col);

printf("\nChosen objects Colum= %d\n", Col);

if (Col <= 0)printf("\nColume of objects must be greater than 0!\n");

} while (Col <= 0);

//khởi tạo ma trận theo kích thước nhập vào

pMatrix = new double[Row \* Col];

pResult = new double[Col];

pSerialResult = new double[Col];

// Definition of matrix and vector elements

DummyDataInitialization(pMatrix, Row, Col);

}

// Function for formatted matrix output

void PrintMatrix(double\* pMatrix, int RowCount, int ColCount) {

int i, j; // Loop variables

for (i = 0; i < RowCount; i++)

{

for (j = 0; j < ColCount; j++)

printf("%7.4f ", pMatrix[i \* ColCount + j]);

printf("\n");

}

}

// Function for formatted vector output

void PrintVector(double\* pVector, int Size) {

int i;

for (i = 0; i < Size; i++)

printf("%7.4f ", pVector[i]);

}

// !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

double SerialResultCalculation(double\* pMatrix, double\*& pSerialResult, int Row, int Col)

{

int i, j;

double result = 0;

for (j = 0; j < Col; j++)

{

pSerialResult[j] = pMatrix[j];

for (i = 0; i < Row; i++)

{

if (pMatrix[i \* Col + j] > pSerialResult[j])

pSerialResult[j] = pMatrix[i \* Col + j];

}

result += pSerialResult[j] / Col;

}

return result;

}

// !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

double ParallelResultCalculation(double\* pMatrix, double\* pResult, int Row, int Col)

{

int i, j;

double sum[NUM\_THREADS] = { 0.0 };

double result = 0.0;

// get largest number of each column, store in pResult

#pragma omp parallel for private(i)

for (j = 0; j < Col; j++)

{

pResult[j] = pMatrix[j];

for (i = 0; i < Row; i++)

{

if (pMatrix[i \* Col + j] > pResult[j])

pResult[j] = pMatrix[i \* Col + j];

}

}

#pragma omp parallel

{

int i, nthreads;

nthreads = omp\_get\_num\_threads();

for (i = omp\_get\_thread\_num(); i < Col; i += nthreads)

sum[omp\_get\_thread\_num()] += pResult[i];

}

for (i = 0; i < NUM\_THREADS; i++)

result += sum[i];

return result / Col;

}

// Parallel Remake

double xParallelResultCalculation(double\* pMatrix, double\* pResult, int Row, int Col)

{

int i, j, nthreads;

double Result[NUM\_THREADS] = { 0.0 };

double result = 0;

#pragma omp parallel private(i, j)

{

nthreads = omp\_get\_num\_threads();

for (j = omp\_get\_thread\_num(); j < Col; j += nthreads)

{

pResult[j] = pMatrix[j];

for (i = 0; i < Row; i++)

{

if (pMatrix[i \* Col + j] > pResult[j])

pResult[j] = pMatrix[i \* Col + j];

}

Result[omp\_get\_thread\_num()] += pResult[j];

}

}

for (i = 0; i < NUM\_THREADS; i++)

{

result += Result[i] / Col;

}

return result;

}

// Function for computational process termination

void ProcessTermination(double\* pMatrix, double\* pResult, double\* pSerialResult) {

delete[] pMatrix;

delete[] pResult;

delete[] pSerialResult;

}

int main() {

/\* DECLARE \*/

double\* pMatrix; // The first argument - initial matrix

double\* pResult; // Result vector for matrix-vector multiplication

//int Size; // Sizes of initial matrix and vector

int Row, Col;

double ParallelResult;

double SerialResult;

double\* pSerialResult;

/\* INIT \*/

omp\_set\_num\_threads(NUM\_THREADS);

printf("Parallel matrix-vector multiplication program\n");

// Memory allocation and definition of objects' elements

ProcessInitialization(pMatrix, pResult, pSerialResult, Row, Col);

// Matrix and vector output

printf("Initial Matrix \n");

//PrintMatrix(pMatrix, Row, Col);

/\* PARALLEL EXECUTION \*/

auto pStart = high\_resolution\_clock::now();

ParallelResult = xParallelResultCalculation(pMatrix, pResult, Row, Col);

auto pFinish = high\_resolution\_clock::now();

auto pDuration = duration\_cast<microseconds>(pFinish - pStart);

/\* TEST RESULT WITH A SERIAL ALGORITHM \*/

auto sStart = high\_resolution\_clock::now();

SerialResult = SerialResultCalculation(pMatrix, pSerialResult, Row, Col);

auto sFinish = high\_resolution\_clock::now();

auto sDuration = duration\_cast<microseconds>(sFinish - sStart);

printf("Serial Result = %lf\n", SerialResult);

printf("Parallel Result = %lf\n", ParallelResult);

/\* PRINT RESULT \*/

//PrintVector(pResult, Col);

//printf("\nResult = %lf\n", ParallelResult);

// Printing the time spent by matrix-vector multiplication

cout << "Time of parallel execution: " << pDuration.count() << " microseconds" << endl;

cout << "Time of serial execution: " << sDuration.count() << " microseconds" << endl;

/\* COMPUTATIONAL PROCESS TERMINATION \*/

ProcessTermination(pMatrix, pResult, pSerialResult);

return 0;

}

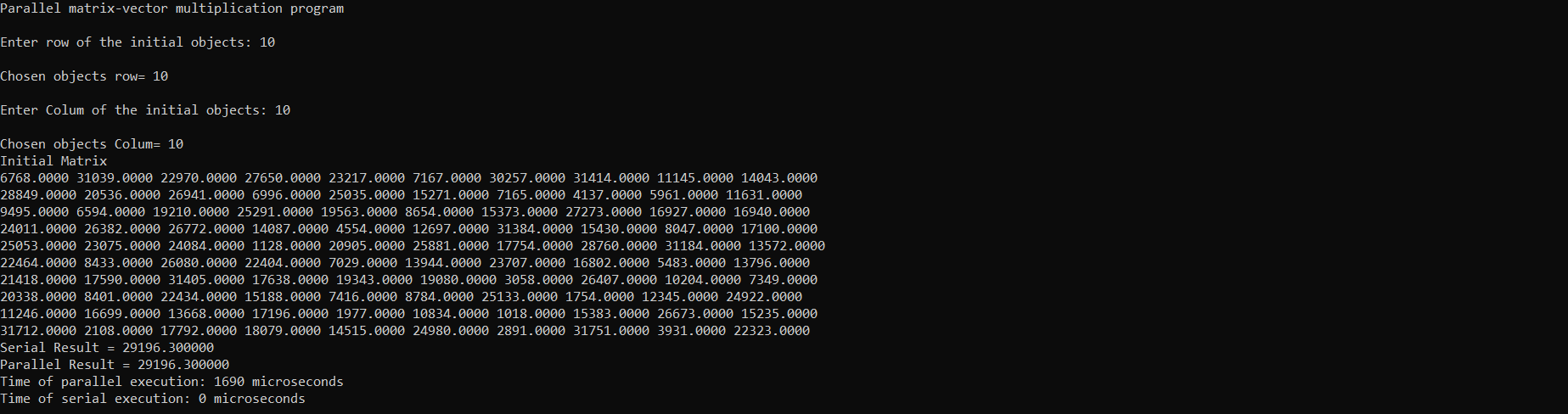
**KẾT QUẢ CHẠY CHƯƠNG TRÌNH**

CÁC TH:

1. T=4 ; R =10 ; C=10 => In MT
2. T=4 ; R =10 ; C=1.000.000
3. T=8 ; R =10; C=1.000.000
4. T=8 ; R=1.000.000; C=10
5. T=16 ; R=1.000.000; C=10
6. T = 16; R = 111; C= 123
7. T=32; R= 1000; C =1000

Tính hiệu suất: gấp bao nhiêu lần

* Bảng thời gian; kích thước; số luồng; hiều suất
  1. Trường hợp 1: T=4 ; R =10 ; C=10
  2. window



* 1. Linux

1. Trường hợp 2: T=4 ; R =10 ; C=1.000.000
   1. window



* 1. Linux

1. Trường hợp 3: T=8 ; R =10 ; C=1.000.000
   1. window
   2. Linux
2. Trường hợp 1: T=8 ; R =1.000.000 ; C=10
   1. window



* 1. Linux

1. Trường hợp 1: T=16 ; R =1.000.000 ; C=10
   1. window



* 1. Linux

1. Trường hợp 1: T=16 ; R =111 ; C=123
   1. window



* 1. Linux

1. Trường hợp 1: T=32 ; R =1000 ; C=1000
   1. window



* 1. Linux

