

TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN TOÁN ỨNG DỤNG VÀ TIN HỌC



BÁO CÁO MÔN
TÍNH TOÁN SONG SONG

Giảng viên hướng dẫn:

TS. Đoàn Duy Trung

Thành viên nhóm: 05

Họ và Tên	MSSV
Nguyễn Hữu Thuật	20185410
Trang Hải Long	20185382
Trần Xuân Hiếu	20185354
Lại Tiến Long	20185376

CHƯƠNG TRÌNH

/*

INPUT: Matrix of m rows, n cols

OUTPUT: The average value of each column's highest value for all columns of the input matrix

Ex: Given a 3x3 matrix such as:

```
[ 100  200  300
  400  200  500
  100  900  100 ]
```

--> the highest value of each column respectively: 400, 900, 500

--> the expected result of the calculation: $(400 + 900 + 500) / 3 = 600$

*/

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include <omp.h>
```

```
#include <chrono>
```

```
#include <iostream>
```

```
using namespace std;
```

```
using namespace std::chrono;
```

```
/*  CONSTANTS  */
```

```
constexpr auto NUM_THREADS = 8;
```

```
/*  FUNCTION DECLARATIONS  */
```

```
// Memory allocation and definition of objects' elements
```

```

void ProcessInitialization(double*& Matrix, double*& tempResults, int& Row,
int& Col);

// Generating random values for the input matrix

void DummyDataInitialization(double* Matrix, int Row, int Col);

// Calculations

double SerialResultCalculation(double* Matrix, double*& tempResults, int Row,
int Col);

double ParallelResultCalculation(double* Matrix, double* tempResults, int
Row, int Col);

// Extra Parallel function using only one parallel code block

double xParallelResultCalculation(double* Matrix, double* tempResults, int
Row, int Col);

// Function for computational process termination

void ProcessTermination(double* Matrix, double* tempResults);


int main() {
    /* DECLARE VARIABLES */

    int Row, Col;           // # of Rows and Columns of the input matrix
    double* Matrix;         // Matrix input
    double* tempResults;    // Vector to store intermediate results
    double ParallelResult;  // result calculated by Parallel algorithm
    double SerialResult;    // Result calculated by Serial algorithm


    /* INIT */

    omp_set_num_threads(NUM_THREADS);
    ProcessInitialization(Matrix, tempResults, Row, Col);


    /* PARALLEL EXECUTION */

    auto pStart = high_resolution_clock::now();

```

```

    ParallelResult = xParallelResultCalculation(Matrix, tempResults, 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(Matrix, tempResults, Row, Col);
    auto sFinish = high_resolution_clock::now();
    auto sDuration = duration_cast<microseconds>(sFinish - sStart);

    /* PRINT RESULT */
    // Printing serial and parallel results
    printf("Serial Result = %lf\n", SerialResult);
    printf("Parallel Result = %lf\n", ParallelResult);
    // Printing the time spent by the calculations
    cout << "Time of parallel execution: " << pDuration.count() << "
microseconds" << endl;
    cout << "Time of serial execution: " << sDuration.count() << "
microseconds" << endl;

    /* COMPUTATIONAL PROCESS TERMINATION */
    ProcessTermination(Matrix, tempResults);
    return 0;
}

/* FUNCTION DEFINITIONS */

void DummyDataInitialization(double* Matrix, int Row, int Col)
{
    int i, j;

```

```

srand(unsigned(clock()));
for (i = 0; i < Row; i++)
{
    for (j = 0; j < Col; j++)
        Matrix[i * Col + j] = 3.0 + rand() % 2000000;    // Using random
number for matrix elements
    }
}

void ProcessInitialization(double*& Matrix, double*& tempResults, int& Row,
int& Col)
{
    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 Column 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);

    // Declare matrix and tempResults array
    Matrix = new double[Row * Col];
    tempResults = new double[Col];

```

```

        // Initialize matrix elements
        DummyDataInitialization(Matrix, Row, Col);
    }

double SerialResultCalculation(double* Matrix, double*& tempResults, int Row,
int Col)
{
    int i, j;
    double result = 0;

    for (j = 0; j < Col; j++)
    {
        tempResults[j] = Matrix[j];
        for (i = 0; i < Row; i++)
        {
            if (Matrix[i * Col + j] > tempResults[j])
                tempResults[j] = Matrix[i * Col + j];
        }
        result += tempResults[j] / Col;
    }

    return result;
}

double ParallelResultCalculation(double* Matrix, double* tempResults, 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 tempResults
#pragma omp parallel for private(i)
    for (j = 0; j < Col; j++)
    {
        tempResults[j] = Matrix[j];
        for (i = 0; i < Row; i++)
        {
            if (Matrix[i * Col + j] > tempResults[j])
                tempResults[j] = Matrix[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()] += tempResults[i];
    }

    for (i = 0; i < NUM_THREADS; i++)
        result += sum[i];

    return result / Col;
}

double xParallelResultCalculation(double* Matrix, double* tempResults, 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)
        {
            tempResults[j] = Matrix[j];
            for (i = 0; i < Row; i++)
            {
                if (Matrix[i * Col + j] > tempResults[j])
                    tempResults[j] = Matrix[i * Col + j];
            }
            Result[omp_get_thread_num()] += tempResults[j];
        }
    }

    for (i = 0; i < NUM_THREADS; i++)
    {
        result += Result[i] / Col;
    }

    return result;
}

void ProcessTermination(double* Matrix, double* tempResults) {
    delete[] Matrix;
}

```

```
delete[] tempResults;

}

}
```

KẾT QUẢ CHẠY CHƯƠNG TRÌNH

I. Chạy chương trình với các trường hợp:

1. Số luồng = 4; Số hàng = 10 ; Số cột =10 (In ma trận ở trường hợp này)
2. Số luồng = 4 ; Số hàng = 10 ; Số cột =1.000.000
3. Số luồng = 8 ; Số hàng = 10; Số cột 1.000.000
4. Số luồng = 8 ; Số hàng = 1.000.000; Số cột =10
5. Số luồng = 16 ; Số hàng =1.000.000; Số cột =10
6. Số luồng = 16; Số hàng = 111; Số cột = 123
7. Số luồng = 32; Số hàng = 1000; Số cột =1000

1. Trường hợp 1: Số luồng = 4; Số hàng = 10 ; Số cột =10

1.1. window

```
Enter row of the initial objects: 10
Chosen objects row= 10
Enter Column of the initial objects: 10
Chosen objects Colum= 10
Initial Matrix
6768.0000 31039.0000 22970.0000 27650.0000 23217.0000 7167.0000 30257.0000 31414.0000 11145.0000 14043.0000
28849.0000 20536.0000 26941.0000 6096.0000 25035.0000 15271.0000 7165.0000 4137.0000 5961.0000 11631.0000
9495.0000 6594.0000 19210.0000 25291.0000 19263.0000 8654.0000 15273.0000 27273.0000 16927.0000 16940.0000
24011.0000 26382.0000 26772.0000 14087.0000 4554.0000 12697.0000 31384.0000 15430.0000 8047.0000 17100.0000
25053.0000 23075.0000 24084.0000 1128.0000 20905.0000 25881.0000 17754.0000 28760.0000 31184.0000 13572.0000
22464.0000 8433.0000 26080.0000 22404.0000 7029.0000 13944.0000 23707.0000 16802.0000 5483.0000 13796.0000
21418.0000 17590.0000 31405.0000 17638.0000 19343.0000 19080.0000 3058.0000 26407.0000 10204.0000 7349.0000
20338.0000 8401.0000 22434.0000 15188.0000 7416.0000 8784.0000 25133.0000 1754.0000 12345.0000 24922.0000
11246.0000 16699.0000 13663.0000 17196.0000 1977.0000 10034.0000 1018.0000 15383.0000 26673.0000 15235.0000
31712.0000 2108.0000 17792.0000 18079.0000 14515.0000 24900.0000 2891.0000 31751.0000 3931.0000 22323.0000
Serial Result = 29196.300000
Parallel Result = 29196.300000
Time of parallel execution: 1690 microseconds
Time of serial execution: 0 microseconds
```

1.2. Linux

```
Enter row of the initial objects: 10
Chosen objects row= 10
Enter Column of the initial objects: 10
Chosen objects Colum= 10
Initial Matrix
1140001.0000 1942642.0000 875654.0000 649030.0000 1503796.0000 155630.0000 438870.0000 92739.0000 1036368.0000 266089.0000
525655.0000 1910417.0000 1500482.0000 1941231.0000 63805.0000 1311236.0000 1294464.0000 1937717.0000 967598.0000 1712908.0000
804233.0000 1936280.0000 1510225.0000 1816769.0000 699927.0000 1978011.0000 1460528.0000 887371.0000 1929321.0000 632243.0000
1140356.0000 1069319.0000 574882.0000 16007.0000 234698.0000 595028.0000 171634.0000 673566.0000 687764.0000 1724351.0000
1456004.0000 1729769.0000 1634765.0000 1472836.0000 1670997.0000 214920.0000 784069.0000 965459.0000 152634.0000 1751664.0000
1194716.0000 1473216.0000 1687941.0000 704938.0000 1289982.0000 904217.0000 1199298.0000 1266860.0000 307937.0000 1644968.0000
1899100.0000 1448290.0000 1230637.0000 990331.0000 1464294.0000 1465332.0000 1585356.0000 152277.0000 655247.0000 789470.0000
1876625.0000 627601.0000 519236.0000 27740.0000 100434.0000 706582.0000 242657.0000 884500.0000 188390.0000 395288.0000
1152514.0000 1383103.0000 384854.0000 840452.0000 604390.0000 1674833.0000 261019.0000 320037.0000 941690.0000 568953.0000
481355.0000 840787.0000 533593.0000 1711989.0000 1831116.0000 514236.0000 1693670.0000 1932821.0000 666511.0000 865267.0000
Serial Result = 1859241.300000
Parallel Result = 1859241.300000
Time of parallel execution: 360 microseconds
Time of serial execution: 3 microseconds
```

2. Trường hợp 2: Số luồng = 4 ; Số hàng = 10 ; Số cột = 1.000.000

2.1. window

```
Enter row of the initial objects: 10
Chosen objects row= 10
Enter Colum of the initial objects: 1000000
Chosen objects Colum= 1000000
Initial Matrix
Serial Result = 29793.563501
Parallel Result = 29793.563501
Time of parallel execution: 27299 microseconds
Time of serial execution: 98298 microseconds
```

2.2. Linux

```
Enter row of the initial objects: 10
Chosen objects row= 10
Enter Colum of the initial objects: 1000000
Chosen objects Colum= 1000000
Serial Result = 1817731.217977
Parallel Result = 1817731.217977
Time of parallel execution: 34556 microseconds
Time of serial execution: 53084 microseconds
```

3. Trường hợp 3: Số luồng = 8 ; Số hàng = 10; C Số cột 1.000.000

3.1. window

```
Enter row of the initial objects: 10
Chosen objects row= 10
Enter Colum of the initial objects: 1000000
Chosen objects Colum= 1000000
Serial Result = 29795.050551
Parallel Result = 29795.050551
Time of parallel execution: 37957 microseconds
Time of serial execution: 90567 microseconds
```

3.2. Linux

```
Enter row of the initial objects: 10
Chosen objects row= 10
Enter Colum of the initial objects: 1000000
Chosen objects Colum= 1000000
Serial Result = 1818040.005720
Parallel Result = 1818040.005720
Time of parallel execution: 19049 microseconds
Time of serial execution: 54412 microseconds
```

4. Trường hợp 4: Số luồng = 8 ; Số hàng = 1.000.000; Số cột = 10

4.1. window

```
Enter row of the initial objects: 1000000
Chosen objects row= 1000000
Enter Colum of the initial objects: 10
Chosen objects Colum= 10
Initial Matrix
Serial Result = 32770.000000
Parallel Result = 32770.000000
Time of parallel execution: 31007 microseconds
Time of serial execution: 98055 microseconds
```

4.2. Linux

```
Enter row of the initial objects: 1000000
Chosen objects row= 1000000
Enter Column of the initial objects: 10
Chosen objects Colum= 10
Serial Result = 2000000.600000
Parallel Result = 2000000.600000
Time of parallel execution: 13819 microseconds
Time of serial execution: 59839 microseconds
```

5. Trường hợp 5: Số luồng = 16 ; Số hàng =1.000.000; Số cột =10

5.1. window

```
Enter row of the initial objects: 1000000
Chosen objects row= 1000000
Enter Column of the initial objects: 10
Chosen objects Colum= 10
Initial Matrix
Serial Result = 32770.000000
Parallel Result = 32770.000000
Time of parallel execution: 27168 microseconds
Time of serial execution: 85703 microseconds
```

5.2. Linux

```
Enter row of the initial objects: 1000000
Chosen objects row= 1000000
Enter Column of the initial objects: 10
Chosen objects Colum= 10
Serial Result = 2000000.900000
Parallel Result = 2000000.900000
Time of parallel execution: 13387 microseconds
Time of serial execution: 59330 microseconds
```

6. Trường hợp 6: Số luồng = 16; Số hàng = 111; Số cột = 123

6.1. window

```
Enter row of the initial objects: 111
Chosen objects row= 111
Enter Column of the initial objects: 123
Chosen objects Colum= 123
Initial Matrix
Serial Result = 32471.000000
Parallel Result = 32471.000000
Time of parallel execution: 8946 microseconds
Time of serial execution: 65 microseconds
```

6.2. Linux

```
Enter row of the initial objects: 111
Chosen objects row= 111
Enter Column of the initial objects: 123
Chosen objects Colum= 123
Serial Result = 1982842.658537
Parallel Result = 1982842.658537
Time of parallel execution: 1009 microseconds
Time of serial execution: 236 microseconds
```

7. Trường hợp 7: Số luồng = 32; Số hàng = 1000; Số cột = 1000

7.1. window

```
Enter row of the initial objects: 1000
Chosen objects row= 1000
Enter Colum of the initial objects: 1000
Chosen objects Colum= 1000
Initial Matrix
Serial Result = 32736.451000
Parallel Result = 32736.451000
Time of parallel execution: 14400 microseconds
Time of serial execution: 8031 microseconds
```

7.2. Linux

```
Enter row of the initial objects: 1000
Chosen objects row= 1000
Enter Colum of the initial objects: 1000
Chosen objects Colum= 1000
Serial Result = 1998024.504000
Parallel Result = 1998024.504000
Time of parallel execution: 3965 microseconds
Time of serial execution: 5668 microseconds
```

II. Thống kê

Bảng số liệu thống kê kết quả khi thực hiện giải thuật tuần tự và song song

STT	Kích thước ma trận		Số luồng (T)	Thời gian thực hiện tuần tự (ms)		Thời gian thực hiện song song (ms)		Hiệu suất (tuần tự/song song)	
	Số hàng (R)	Số cột (C)		Win	Linux	Win	Linux	Win	Linux
Test 1	10	10	4	0	3	1690	360	0.0000	0.0021
Test 2	10	1.000.000	4	98298	53004	27299	34556	0.9002	0.3882
Test 3	10	1.000.000	8	90567	54412	37957	19049	0.2982	0.3570
Test 4	1.000.000	10	8	98055	59839	31007	13819	0.3953	0.5413
Test 5	1.000.000	10	16	85703	59330	27168	13387	0.1971	0.2770
Test 6	111	123	16	65	236	8946	1609	0.0005	0.0092
Test 7	1000	1000	32	8031	5668	14400	3965	0.0174	0.0447

III. Nhận xét

- Khi tăng số lượng luồng xử lý, thời gian được xử lý giảm đi
- Thời gian chạy ở quá trình song song ít hơn ở quá trình tuần tự. Tuy nhiên khi kích thước ma trận nhỏ thì quá trình chạy song song mất nhiều thời gian hơn quá trình chạy tuần tự (bởi vì thời gian chờ trong quá trình chạy song song lâu hơn quá trình tính toán trong quá trình chạy tuần tự).
- Chương trình thực hiện tính toán song song theo cột, nên với ma trận có cùng số phần tử (tích Row*Column bằng nhau) thì ma trận có nhiều hàng hơn sẽ xử lý lâu hơn vì quá trình duyệt các phần tử trong từng cột là duyệt tuần tự.

STT	Kích thước ma trận		Thời gian thực hiện tuần tự (ms)		Thời gian thực hiện song song (ms)		Hiệu suất (tuần tự/song song)	
			Win	Linux	Win	Linux	Win	Linux
Test 1	10	10	0	3	1690	360	0.0000	0.0021
Test 2	10	1.000.000	98298	53004	27299	34556	0.9002	0.3882
Test 3	10	1.000.000	90567	54412	37957	19049	0.2982	0.3570
Test 4	1.000.000	10	98055	59839	31007	13819	0.3953	0.5413
Test 5	1.000.000	10	85703	59330	27168	13387	0.1971	0.2770
Test 6	111	123	65	236	8946	1609	0.0005	0.0092
Test 7	1000	1000	8031	5668	14400	3965	0.0174	0.0447

IV. Báo cáo bài toán ma trận nhân vector

STT	Kích thước ma trận	Thời gian thực hiện tuần tự	Thời gian thực hiện song song	Hiệu suất
Test № 1	10	0.000001	0.004443	2.81341E-05
Test № 2	100	0.000029	0.003529	0.001027203
Test № 3	1000	0.004089	0.005657	0.09035266
Test № 4	2000	0.020137	0.007410	0.339692982
Test № 5	3000	0.067213	0.014322	0.586623726
Test № 6	4000	0.079390	0.038741	0.256156269
Test № 7	5000	0.151565	0.063098	0.30025714
Test № 8	6000	0.217916	0.109199	0.249448255
Test № 9	7000	0.148064	0.081744	0.226414171
Test № 10	8000	0.224541	0.074702	0.375727892
Test № 11	9000	0.357941	0.088998	0.502737421
Test № 12	10 000	0.350015	0.118220	0.370088606