**TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI**

**VIỆN TOÁN ỨNG DỤNG VÀ TIN HỌC**

**---🙢🙠---**



**BÁO CÁO GIỮA KỲ MÔN**

**TÍNH TOÁN SONG SONG**

**Giảng viên hướng dẫn: TS. Vũ Thành Nam**

**Nhóm sinh viên thực hiện:** Nhóm 11 – Chủ đề 11

|  |  |
| --- | --- |
| **Họ tên** | **MSSV** |
| **Nguyễn Thị Bích Ngọc** | **20185388** |
| **Nguyễn Bùi Nam Trường** | **20185418** |
| **Hoàng Huy Hoàng** | **20195877** |

**Mã học phần: MI4364**

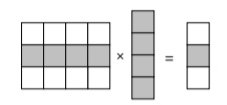
**Mã lớp học: 146169**

**HÀ NỘI, THÁNG 11/2023**

# PHẦN 1: Bài thực hành chung

## Bài toán

*Bài toán*: nhân ma trận A cỡ mxn và vecto b cỡ nx1 kết quả nhận được là 1 vecto cỡ mx1, tức là:



Hình 1: Kết quả nhân ma trận với vecto

*Ví dụ*: nhân ma trận A3x4 và vecto b4x1 ta được vecto c3x1 như sau:

Hình 2: Nhân ma trận và vecto

## Code

#include <cstdio>

#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

#include <windows.h>

#include <time.h>

#include <iostream>

#include <iomanip>

#include <omp.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* SUPPORT METHODs \*/

// Function that converts numbers form LongInt type to double type

double LiToDouble(*LARGE\_INTEGER* *x*) {

    double result = ((double)*x*.HighPart) \* 4.294967296E9 + (double)((*x*).LowPart);

    return result;

}

// Function that gets the timestamp in seconds

double GetTime() {

*LARGE\_INTEGER* lpFrequency, lpPerfomanceCount;

    QueryPerformanceFrequency(&lpFrequency);

    QueryPerformanceCounter(&lpPerfomanceCount);

    return LiToDouble(lpPerfomanceCount) / LiToDouble(lpFrequency);

}

// Function for simple definition of matrix and vector elements

void DummyDataInitialization(double\*& *pMatrix*, double\*& *pResult*, int& *size*) {

    srand(unsigned(clock()));

    for (int i = 0; i < *size*; i++) {

*pResult*[i] = 0.0;

        for (int j = 0; j < *size*; j++) *pMatrix*[i \* *size* + j] = rand() / double(1000);

    }

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* PROCESS METHODs \*/

// Function for memory allocation and definition of object's elements

void ProcessInitialization(double\*& *pMatrix*, double\*& *pResult*, double& *pAvgEven*, double& *pAvgOdd*, int& *size*) {

*pMatrix* = **new** double[*size* \* *size*];

*pResult* = **new** double[*size*];

    DummyDataInitialization(*pMatrix*, *pResult*, *size*);

*pAvgEven* = 0.0;

*pAvgOdd* = 0.0;

}

// Function for serial caculation

void ProcessSerialCalculation(double\* *pMatrix*, double\* *pResult*, double& *pAvgEven*, double& *pAvgOdd*, int& *size*) {

    for (int i = 0; i < *size*; i++) {

*pResult*[i] = *pMatrix*[i \* *size*];

        for (int j = 0; j < *size*; j++) {

            if (i % 2) {

                if (*pResult*[i] > *pMatrix*[i \* *size* + j]) *pResult*[i] = *pMatrix*[i \* *size* + j];

            } else {

                if (*pResult*[i] < *pMatrix*[i \* *size* + j]) *pResult*[i] = *pMatrix*[i \* *size* + j];

            }

        }

    }

    int countEven = 0, countOdd = 0;

    double sumEven = 0, sumOdd = 0;

    for (int i = 0; i < *size*; i++) {

        if (i % 2) {

            sumEven += *pResult*[i]; countEven++;

        } else {

            sumOdd += *pResult*[i]; countOdd++;

        }

    }

*pAvgEven* = sumEven / countEven;

*pAvgOdd* = sumOdd / countOdd;

}

// Function for parallel caculation

void ProcessParalleCalculation(double\* *pMatrix*, double\* *pResult*, double& *pAvgEven*, double& *pAvgOdd*, int& *size*) {

    #pragma *omp* *parallel* *for*

        for (int i = 0; i < *size*; i++) {

*pResult*[i] = *pMatrix*[i \* *size*];

            for (int j = 0; j < *size*; j++) {

                if (i % 2) {

                    if (*pResult*[i] > *pMatrix*[i \* *size* + j]) *pResult*[i] = *pMatrix*[i \* *size* + j];

                } else {

                    if (*pResult*[i] < *pMatrix*[i \* *size* + j]) *pResult*[i] = *pMatrix*[i \* *size* + j];

                }

            }

        }

    int countEven = 0, countOdd = 0;

    double sumEven = 0, sumOdd = 0;

    #pragma *omp* *parallel* *sections*

    {

        #pragma *omp* *section*

        {

            for (int i = 0; i < *size*; i += 2) {

                sumEven += *pResult*[i]; countEven++;

            }

*pAvgEven* = sumEven / countEven;

        }

        #pragma *omp* *section*

        {

            for (int i = 1; i < *size*; i += 2) {

                sumOdd += *pResult*[i]; countOdd++;

            }

*pAvgOdd* = sumOdd / countOdd;

        }

    }

}

// Function for computational process termination

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

**delete[]** *pMatrix*;

**delete[]** *pResult*;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* PROGRAM METHODs \*/

double getDurationProcessSerialCalculation(double\* *pMatrix*, double\* *pResult*, double& *pAvgEven*, double& *pAvgOdd*, int& *size*) {

    // Cấp phát bộ nhớ cho dữ liệu

    ProcessInitialization(*pMatrix*, *pResult*, *pAvgEven*, *pAvgOdd*, *size*);

    // Thực hiện tính toán song song

    double start = GetTime();

    ProcessSerialCalculation(*pMatrix*, *pResult*, *pAvgEven*, *pAvgOdd*, *size*);

    double finish = GetTime();

    // Xóa bộ nhớ dữ liệu

    ProcessTermination(*pMatrix*, *pResult*);

    // Trả về thời gian tính toán

    return finish - start;

}

double getDurationProcessParalleCalculation(double\* *pMatrix*, double\* *pResult*, double& *pAvgEven*, double& *pAvgOdd*, int& *size*) {

    // Cấp phát bộ nhớ cho dữ liệu

    ProcessInitialization(*pMatrix*, *pResult*, *pAvgEven*, *pAvgOdd*, *size*);

    // Thực hiện tính toán song song

    double start = GetTime();

    ProcessParalleCalculation(*pMatrix*, *pResult*, *pAvgEven*, *pAvgOdd*, *size*);

    double finish = GetTime();

    // Xóa bộ nhớ dữ liệu

    ProcessTermination(*pMatrix*, *pResult*);

    // Trả về thời gian tính toán

    return finish - start;

}

int main() {

    double\* pMatrix; // Ma trận ban đầu

    double\* pResult; // Mảng các số cần tìm tại mỗi hàng

    double pAvgEven; // Giá trị trung bình số nhỏ nhất hàng chẵn

    double pAvgOdd; // Giá trị trung bình số lớn nhaasrt hàng lẻ

    int numThreads = 8;

    omp\_set\_num\_threads(numThreads);

    int sizeData[12] = {10, 100, 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000};

    int sizeLength = sizeof(sizeData) / sizeof(sizeData[0]);

    std::cout << std::fixed << std::left << std::setprecision(15);

    std::cout << std::setw(20)

        << "Kích thước ma trận\t"

        << std::setw(30)

        << "Thời gian thực hiện tuần tự\t"

        << std::setw(30)

        << "Thời gian thực hiện song song\t"

        << std::setw(30)

        << "Hiệu suất\t"

        << std::endl;

    for (int i = 0; i < sizeLength; i++) {

        int size = sizeData[i];

        double serialTime = getDurationProcessSerialCalculation(pMatrix, pResult, pAvgEven, pAvgOdd, size);

        double paralleTime = getDurationProcessParalleCalculation(pMatrix, pResult, pAvgEven, pAvgOdd, size);

        double performance = serialTime / (paralleTime \* numThreads);

        std::cout << std::setw(20)

            << sizeData[i] << "\t"

            << std::setw(30)

            << serialTime << "\t"

            << std::setw(30)

            << paralleTime << "\t"

            << std::setw(30)

            << performance << "\t"

            << std::endl;

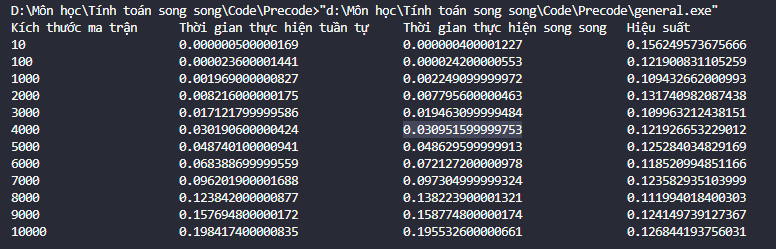
    }

    return 0;

}

## Kết quả bài toán

Hiệu suất:



Hình 3: Kết quả chạy chương trình nhân matrix với vector.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| STT | Kích thước ma trận | Thời gian thực hiện tuần tự | Thời gian thực hiện song song () | Hiệu suất |
| 1 | 10 | 0.000000500000169 | 0.000000400001227 | 0.156249573675666 |
| 2 | 100 | 0.000023600001441 | 0.000024200000553 | 0.121900831105259 |
| 3 | 1000 | 0.001969000000827 | 0.002249099999972 | 0.109432662000993 |
| 4 | 2000 | 0.008216000000175 | 0.007795600000463 | 0.131740982087438 |
| 5 | 3000 | 0.017121799999586 | 0.019463099999484 | 0.109963212438151 |
| 6 | 4000 | 0.030190600000424 | 0.030951599999753 | 0.121926653229012 |
| 7 | 5000 | 0.048740100000941 | 0.048629599999913 | 0.125284034829169 |
| 8 | 6000 | 0.068388699999559 | 0.072127200000978 | 0.118520994851166 |
| 9 | 7000 | 0.096201900001688 | 0.097304999999324 | 0.123582935103999 |
| 10 | 8000 | 0.123842000000877 | 0.138223900001321 | 0.111994018400303 |
| 11 | 9000 | 0.157694800000172 | 0.158774800000174 | 0.124149739127367 |
| 12 | 10000 | 0.198417400000835 | 0.195532600000661 | 0.126844193756031 |

Bảng 1: Kết quả chạy chương trình nhân matrix với vector.

*Nhận xét:*Từ kết quả trên, ta thấy rằng với bộ dữ liệu có kích thước nhỏ thì lập trình song song lâu hơn là lập trình tuần tự, nhưng khi kích thước bộ dữ liệu tăng dần (đến khoảng cỡ 5000x5000) thì lập trình song song lại có thời gian ngắn hơn, nên hiệu quả hơn.

# PHẦN 2: Bài tập nhóm

## Bài toán

*Đề 11:* Viết chương trình tính giá trị trung bình của các phần tử của mảng 2 chiều thỏa mãn điều kiện sau:

a. Phần tử lớn nhất trên hàng lẻ (1,3,5…)

b. Phần tử nhỏ nhất trên hàng chẵn (0, 2, 4, 6…)

(Dùng hàm thời gian so sánh với lập trình tuần tự)

## Code

#include <cstdio>

#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

#include <windows.h>

#include <time.h>

#include <iostream>

#include <iomanip>

#include <omp.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* SUPPORT METHODs \*/

// Function that converts numbers form LongInt type to double type

double LiToDouble(*LARGE\_INTEGER* *x*) {

    double result = ((double)x.HighPart) \* 4.294967296E9 + (double)((x).LowPart);

    return result;

}

// Function that gets the timestamp in seconds

double GetTime() {

    LARGE\_INTEGER lpFrequency, lpPerfomanceCount;

    QueryPerformanceFrequency(&lpFrequency);

    QueryPerformanceCounter(&lpPerfomanceCount);

    return LiToDouble(lpPerfomanceCount) / LiToDouble(lpFrequency);

}

// Function for simple definition of matrix and vector elements

void DummyDataInitialization(double\* *pMatrix*, double\* *pVector*, int *Size*) {

    srand(unsigned(clock()));

    int i, j;

    for (i = 0; i < Size; i++) {

        pVector[i] = rand() / double(1000);

        for (j = 0; j < Size; j++)

            pMatrix[i \* Size + j] = rand() / double(1000);

    }

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* PROCESS METHODs \*/

// Function for memory allocation and definition of object's elements

void ProcessInitialization(double\*& *pMatrix*, double\*& *pVector*, double\*& *pResult*, int& *Size*) {

    pMatrix = **new** double[Size \* Size];

    pVector = **new** double[Size];

    pResult = **new** double[Size];

    DummyDataInitialization(pMatrix, pVector, Size);

}

// Function for serial matrix-vector multiplication

void ProcessSerialCalculation(double\* *pMatrix*, double\* *pVector*, double\* *pResult*, int *Size*) {

    int i, j;

    for (i = 0; i < Size; i++) {

        for (j = 0; j < Size; j++)

            pResult[i] += pMatrix[i \* Size + j] \* pVector[j];

    }

}

// Function for parallel matrix-vector multiplication

void ProcessParalleCalculation(double\* *pMatrix*, double\* *pVector*, double\* *pResult*, int *Size*) {

    int i, j;

    #pragma *omp* *paralell* *for* *private* (*j*)

    for (i = 0; i < Size; i++) {

        for (j = 0; j < Size; j++)

            pResult[i] += pMatrix[i \* Size + j] \* pVector[j];

    }

}

// Function for computational process termination

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

    delete[] pMatrix;

    delete[] pVector;

    delete[] pResult;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* PROGRAM METHODs \*/

double getDurationProcessSerialCalculation(double\* *pMatrix*, double\* *pVector*, double\* *pResult*, int *Size*) {

    // Cấp phát bộ nhớ cho dữ liệu

    ProcessInitialization(pMatrix, pVector, pResult, Size);

    // Thực hiện tính toán song song

    double start = GetTime();

    ProcessSerialCalculation(pMatrix, pVector, pResult, Size);

    double finish = GetTime();

    // Xóa bộ nhớ dữ liệu

    ProcessTermination(pMatrix, pVector, pResult);

    // Trả về thời gian tính toán

    return finish - start;

}

double getDurationProcessParalleCalculation(double\* *pMatrix*, double\* *pVector*, double\* *pResult*, int *Size*) {

    // Cấp phát bộ nhớ cho dữ liệu

    ProcessInitialization(pMatrix, pVector, pResult, Size);

    double start = GetTime();

    ProcessParalleCalculation(pMatrix, pVector, pResult, Size);

    double finish = GetTime();

    // Xóa bộ nhớ dữ liệu

    ProcessTermination(pMatrix, pVector, pResult);

    // Trả về thời gian tính toán

    return finish - start;

}

int main() {

    double\* pMatrix; // Ma trận ban đầu

    double\* pVector; // Vector ban đầu

    double\* pResult; // Kết quả của phép nhân ma trận-vector

    int numThreads = 8;

    omp\_set\_num\_threads(numThreads);

    int sizeData[12] = {10, 100, 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000};

    int sizeLength = sizeof(sizeData) / sizeof(sizeData[0]);

    std::cout << std::fixed << std::left << std::setprecision(15);

    std::cout << std::setw(20)

        << "Kích thước ma trận\t"

        << std::setw(30)

        << "Thời gian thực hiện tuần tự\t"

        << std::setw(30)

        << "Thời gian thực hiện song song\t"

        << std::setw(30)

        << "Hiệu suất\t"

        << std::endl;

    for (int i = 0; i < sizeLength; i++) {

        int size = sizeData[i];

        double serialTime = getDurationProcessSerialCalculation(pMatrix, pVector, pResult, size);

        double paralleTime = getDurationProcessParalleCalculation(pMatrix, pVector, pResult, size);

        double performance = serialTime / (paralleTime \* numThreads);

        std::cout << std::setw(20)

            << sizeData[i] << "\t"

            << std::setw(30)

            << serialTime << "\t"

            << std::setw(30)

            << paralleTime << "\t"

            << std::setw(30)

            << performance << "\t"

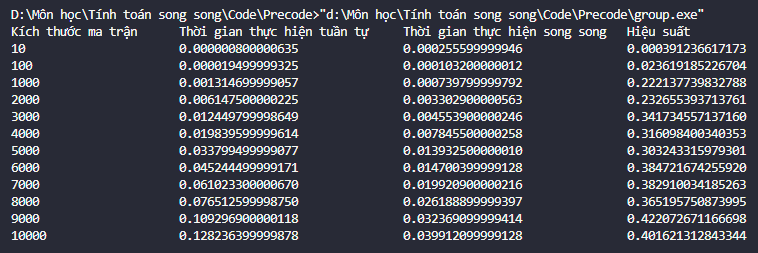
            << std::endl;

    }

    return 0;

}

## Kết quả bài toán



Hình 3: Kết quả chạy chương trình tính trung bình.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| STT | Kích thước ma trận | Thời gian thực hiện tuần tự | Thời gian thực hiện song song () | Hiệu suất |
| 1 | 10 | 0.000000800000635 | 0.000255599999946 | 0.000391236617173 |
| 2 | 100 | 0.000019499999325 | 0.000103200000012 | 0.023619185226704 |
| 3 | 1000 | 0.001314699999057 | 0.000739799999792 | 0.222137739832788 |
| 4 | 2000 | 0.006147500000225 | 0.003302900000563 | 0.232655393713761 |
| 5 | 3000 | 0.012449799998649 | 0.004553900000246 | 0.341734557137160 |
| 6 | 4000 | 0.019839599999614 | 0.007845500000258 | 0.316098400340353 |
| 7 | 5000 | 0.033799499999077 | 0.013932500000010 | 0.303243315979301 |
| 8 | 6000 | 0.045244499999171 | 0.014700399999128 | 0.384721674255920 |
| 9 | 7000 | 0.061023300000670 | 0.019920900000216 | 0.382910034185263 |
| 10 | 8000 | 0.076512599998750 | 0.026188899999397 | 0.365195750873995 |
| 11 | 9000 | 0.109296900000118 | 0.032369099999414 | 0.422072671166698 |
| 12 | 10000 | 0.128236399999878 | 0.039912099999128 | 0.401621312843344 |

Bảng 2: Kết quả chạy chương trình tính trung bình.

*Nhận xét:*Từ kết quả trên, ta thấy rằng với bộ dữ liệu có kích thước nhỏ thì lập trình song song lâu hơn là lập trình tuần tự, nhưng khi kích thước bộ dữ liệu tăng dần (đến khoảng cỡ 1000x1000) thì lập trình song song lại có thời gian ngắn hơn, nên hiệu quả hơn.