Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования «Кубанский государственный университет»

Кафедра вычислительных технологий

**ОТЧЕТ**

о выполнении лабораторной работы № 1

по дисциплине «Теория параллельных алгоритмов»

Выполнил:

Корнилов К.А.

Проверил: преподаватель

Нигодин Е. А.

Краснодар

2024

**Тема работы:** Умножение матриц.

**Цель:** Распараллеливание процесса перемножения матриц с использованием технологии OpenMP.

**Задание:**

1. Составить последовательный алгоритм. Построить граф алгоритма. Исследовать на нём зависимости и возможности распараллеливания.
2. Выполнить параллельную реализацию для случаев ленточного и блочного разделения матриц.
3. Составить таблицу, отражающую сравнительное время выполнения последовательного и параллельных алгоритмов для разных размеров матриц на 2-х и 4-х ядрах. (+ 3 ядра, + график в Excel).

**Ход работы:**

1. Был составлен и написан на языке С++ последовательный алгоритм перемножения матриц.

Входными данными алгоритма выступают: матрица A размерности и матрица B размерности .

Результатом выступает матрица C размерности , полученная путём перемножения A на B.

Получение результата работы алгоритма возможно лишь в том случае, если .

Элементы искомой матрицы C вычисляются следующим образом:

Последовательный алгоритм перемножения матриц схематично изображен на рисунке 1 для частного случая (матрицы 3x4 и 4x3):

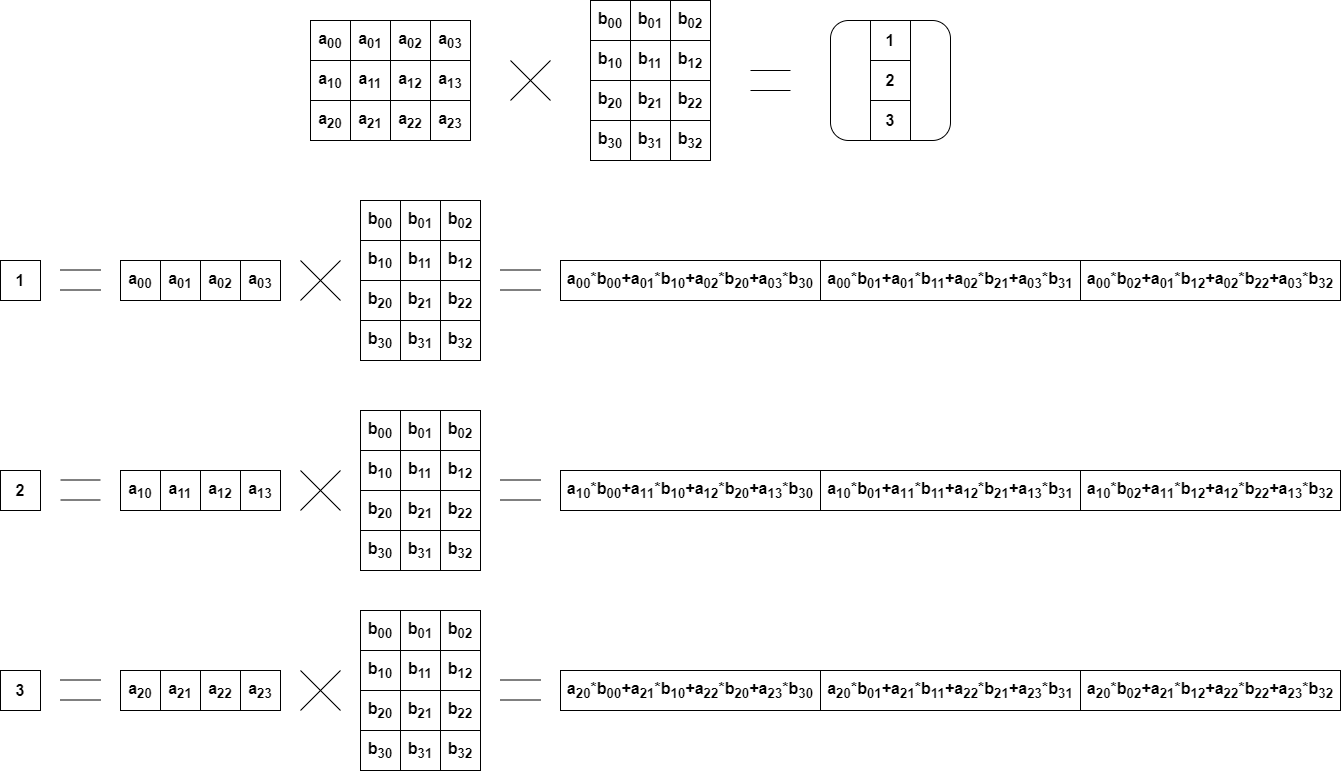


Рисунок 1 – Схема последовательного алгоритма умножения матриц

Код реализованного алгоритма имеет следующий вид:

double\*\* seq\_matr\_multip(double\*\*matr1,int N1,int M1,double\*\*matr2,int N2,int M2) {

double\*\* result = new double\* [N1];

for (int i= 0; i < N1; i++) {

result[i] = new double[M2];

}

for (int i = 0; i < N1; i++) {

for (int j = 0; j < M2; j++) {

double s = 0;

for (int m = 0; m < M1; m++) {

s = s + matr1[i][m] \* matr2[m][j];

}

result[i][j] = s;

}

}

return result;

}

Согласно схеме алгоритма первой возможностью распараллеливания является распараллеливание умножения строк матрицы А на матрицу B. Данные перемножения независимы друг от друга и поэтому могут быть выполнены параллельно и затем объединены в одну результирующую матрицу. Схема алгоритма параллельного ленточного умножения матриц представлена на рисунке 2 для частного случая.

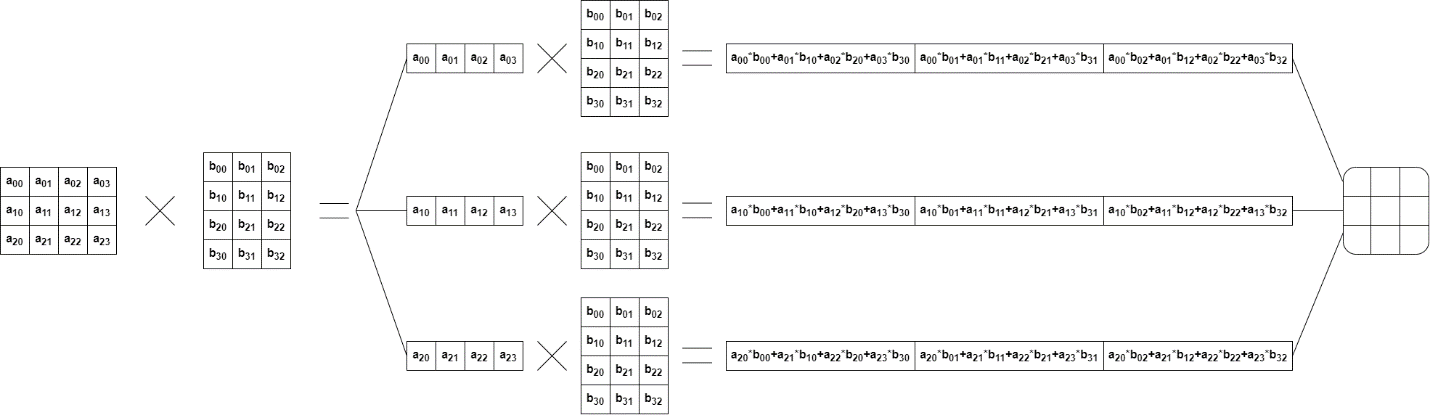


Рисунок 2 – Схема первого алгоритма параллельного ленточного умножения матриц

Кроме того, можно таким же образом поступить с матрицей B – разделим её на вертикальные ленты, вычисляя таким образом в каждом потоке только один элемент искомой матрицы, после чего мы объединим их. Схема второго варианта алгоритма параллельного ленточного умножения матриц для частного случая представлена на рисунке 3.

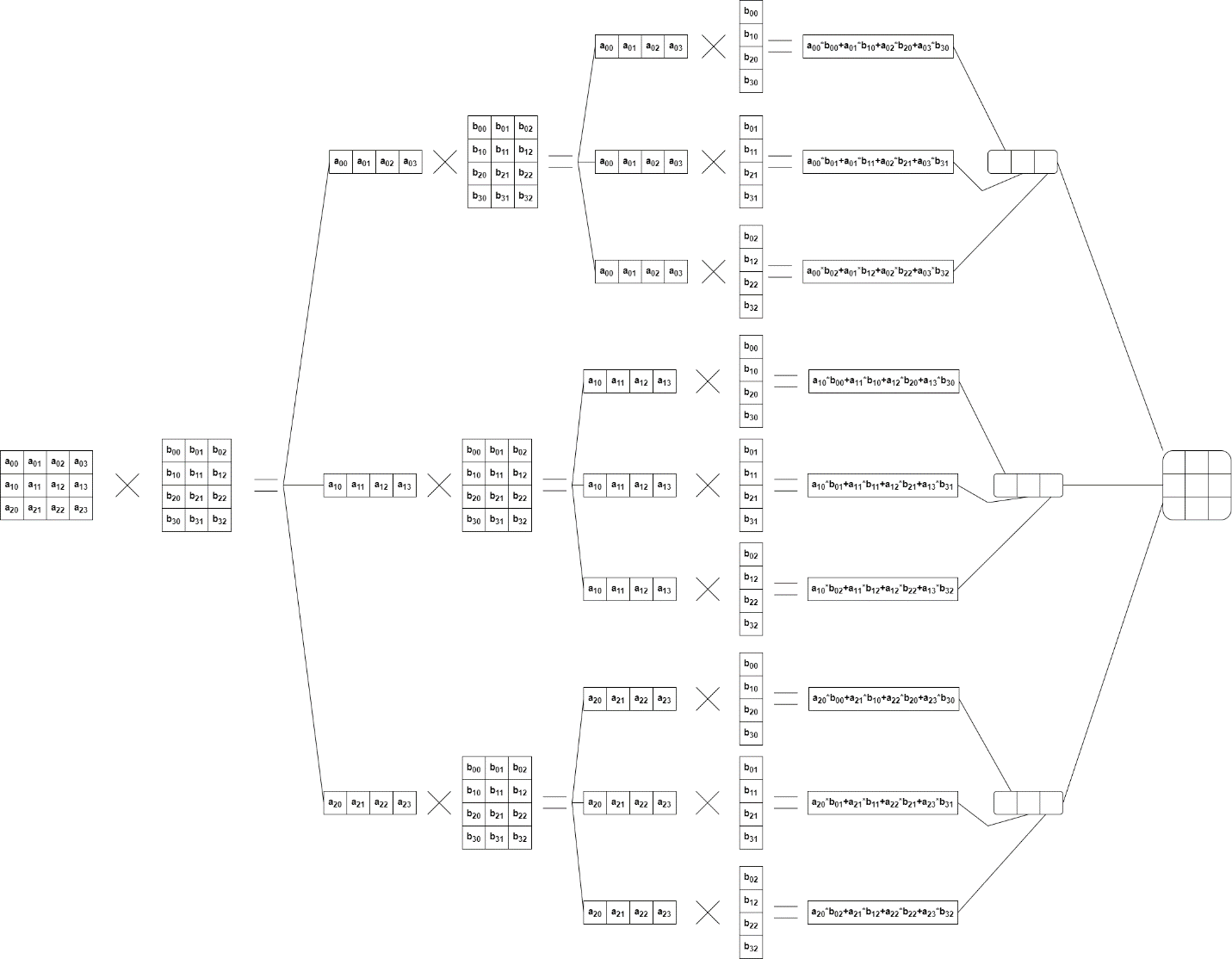


Рисунок 3 – Схема второго варианта параллельного алгоритма ленточного умножения матриц

Помимо рассмотрения матриц в виде наборов строк и столбцов, также можно использовать блочное представление матриц. В этом случае умножение матриц A и B представляется в виде пошагового умножения матриц меньшего размера. Данные умножения также могут быть выполнены параллельно. Частичная схема алгоритма параллельного блочного умножения матриц для частного случая представлена на рисунке 4.

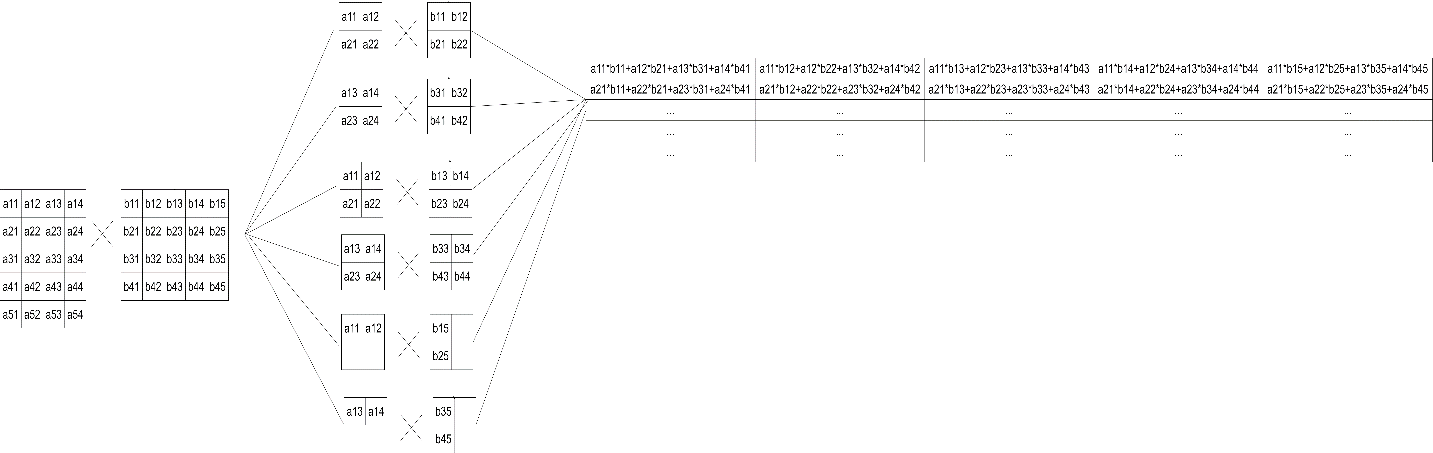


Рисунок 4 – Схема алгоритма параллельного блочного умножения матриц

1. Была написана параллельная реализация ленточного и блочных умножений матриц на языке С++ с использованием технологии OpenMP.

Код параллельной реализации ленточного умножения матриц:

double\*\* lent\_mult(double\*\* matr1, double\*\* matr2, int N1, int M1, int N2, int M2) {

double\*\* result = new double\* [N1];

for (int i = 0; i < N1; i++) {

result[i] = new double[M2];

for (int j = 0; j < M2; j++) {

result[i][j] = 0;

}

}

int n;

cout << "Введите количество нитей" << endl;

cin >> n;

omp\_set\_num\_threads(n);

#pragma omp parallel for collapse(3) num\_threads(n)

for (int i = 0; i < N1; i++) {

for (int j = 0; j < M2; j++) {

for (int k = 0; k < M1; k++) {

result[i][j] = result[i][j] + matr1[i][k] \* matr2[k][j];

}

}

}

return result;

}

Код параллельной реализации блочного умножения матриц:

double\*\* block\_mult(double\*\* matr1, double\*\* matr2, int N1, int M1, int N2, int M2) {

double\*\* result = new double\* [N1];

for (int i = 0; i < N1; i++) {

result[i] = new double[M2];

for (int j = 0; j < M2; j++) {

result[i][j] = 0;

}

}

int n;

cout << "Введите количество нитей" << endl;

cin >> n;

omp\_set\_num\_threads(n);

int blockSize = max(M1 / n, 1);

cout << "Размер блока: " << blockSize << endl;

#pragma omp parallel for collapse(3) num\_threads(n)

for (int i = 0; i < N1; i += blockSize) {

for (int j = 0; j < M2; j += blockSize) {

for (int k = 0; k < M1; k += blockSize) {

// Умножение блоков матриц

for (int ii = i; ii < min(i + blockSize, N1); ii++) {

for (int jj = j; jj < min(j + blockSize, M2); jj++) {

for (int kk = k; kk < min(k + blockSize, M1); kk++) {

result[ii][jj] += matr1[ii][kk] \* matr2[kk][jj];

}

}

}

print\_matr(result, N1, M2);

cout << "---" << endl;

}

}

}

return result;

}

1. Используя написанные программы, была проведена оценка времени работы алгоритмов для различных размеров матриц и для различного количество ядер. Результаты выполненных оценок представлены в виде графика времени выполнения на рисунке 5.

По оси Ox изображены размеры перемножаемых матриц. По оси Oy изображено время работы алгоритма.

Черным цветом на графике изображена время выполнения для последовательного алгоритма умножения матриц.

Оранжевым цветом на графике изображена время выполнения для алгоритма параллельного ленточного умножения матриц на 2 потоках.

Красным цветом на графике изображена время выполнения для алгоритма параллельного ленточного умножения матриц на 3 потоках.

Темно-красным цветом на графике изображена время выполнения для алгоритма параллельного ленточного умножения матриц на 4 потоках.

Светло-зеленым цветом на графике изображена время выполнения для алгоритма параллельного блочного умножения матриц на 2 потоках.

Зеленым цветом на графике изображена время выполнения для алгоритма параллельного блочного умножения матриц на 3 потоках.

Темно-зеленым цветом на графике изображена время выполнения для алгоритма параллельного блочного умножения матриц на 4 потоках.

Рисунок 5 – График зависимости времени работы программы от размера матрицы и количества ядер

**Вывод:** Было осуществлено распараллеливание процесса перемножения матриц с использованием технологии OpenMP и была осуществлена оценка времени работы данной программы. Согласно проведенными исследованиям максимальная скорость работы программы была достигнута на 4 потоках.

**ПРИЛОЖЕНИЕ.**

**Приложение 1. Полный код программы**

#include <iostream>

using namespace std;

#include<string>

#include<sstream>

#include<vector>

#include<omp.h>

#include<cmath>

#include<ctime>

double\*\* input\_matr(int N, int M) {

double\*\* matr = new double\* [N];

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

matr[i] = new double[M];

}

for (int i = 0; i < N + 1; i++) {

string inp = "";

getline(std::cin, inp);

stringstream ss(inp);

int j = 0;

string word;

while (ss >> word) {

matr[i - 1][j] = stod(word);

j = j + 1;

}

}

return matr;

}

void print\_matr(double\*\* matr, int N, int M) {

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

for (int j = 0; j < M; j++) {

cout << matr[i][j] << ' ';

}

cout << endl;

}

}

double\*\* seq\_matr\_multip(double\*\* matr1, int N1, int M1, double\*\* matr2, int N2, int M2) {

double\*\* result = new double\* [N1];

for (int i = 0; i < N1; i++) {

result[i] = new double[M2];

}

for (int i = 0; i < N1; i++) {

for (int j = 0; j < M2; j++) {

double s = 0;

for (int m = 0; m < M1; m++) {

s = s + matr1[i][m] \* matr2[m][j];

}

result[i][j] = s;

}

}

return result;

}

double\*\* lent\_mult(double\*\* matr1, double\*\* matr2, int N1, int M1, int N2, int M2, int threads) {

double\*\* result = new double\* [N1];

for (int i = 0; i < N1; i++) {

result[i] = new double[M2];

for (int j = 0; j < M2; j++) {

result[i][j] = 0;

}

}

omp\_set\_num\_threads(threads);

#pragma omp parallel for collapse(3) num\_threads(threads)

for (int i = 0; i < N1; i++) {

for (int j = 0; j < M2; j++) {

for (int k = 0; k < M1; k++) {

result[i][j] = result[i][j] + matr1[i][k] \* matr2[k][j];

}

}

}

return result;

}

double\*\* block\_mult(double\*\* matr1, double\*\* matr2, int N1, int M1, int N2, int M2,int threads) {

double\*\* result = new double\* [N1];

for (int i = 0; i < N1; i++) {

result[i] = new double[M2];

for (int j = 0; j < M2; j++) {

result[i][j] = 0;

}

}

omp\_set\_num\_threads(threads);

int blockSize = max(M1 / threads, 1);

cout << "Размер блока: " << blockSize << endl;

#pragma omp parallel for collapse(3) num\_threads(threads)

for (int i = 0; i < N1; i += blockSize) {

for (int j = 0; j < M2; j += blockSize) {

for (int k = 0; k < M1; k += blockSize) {

// Умножение блоков матриц

for (int ii = i; ii < min(i + blockSize, N1); ii++) {

for (int jj = j; jj < min(j + blockSize, M2); jj++) {

for (int kk = k; kk < min(k + blockSize, M1); kk++) {

result[ii][jj] += matr1[ii][kk] \* matr2[kk][jj];

}

}

}

}

}

}

return result;

}

int main()

{

setlocale(LC\_ALL, "russian");

int N1, M1;

cin >> N1 >> M1;

double\*\* matr1 = input\_matr(N1, M1);

int N2, M2;

cin >> N2 >> M2;

double\*\* matr2 = input\_matr(N2, M2);

double time = clock();

double\*\*result=seq\_matr\_multip(matr1, N1, M1, matr2, N2, M2);

double end\_time = clock();

cout << "Result time: " << double((end\_time - time)) << endl;

print\_matr(result, N1, M2);

cout << "Блок 2 нить" << endl;

time = clock();

result = block\_mult(matr1, matr2, N1, M1, N2, M2,2);

end\_time = clock();

cout << "Result time: " << double((end\_time - time)) << endl;

print\_matr(result, N1, M2);

cout << "Блок 3 нить" << endl;

time = clock();

result = block\_mult(matr1, matr2, N1, M1, N2, M2,3);

end\_time = clock();

cout << "Result time: " << double((end\_time - time)) << endl;

print\_matr(result, N1, M2);

cout << "Блок 4 нить" << endl;

time = clock();

result = block\_mult(matr1, matr2, N1, M1, N2, M2,4);

end\_time = clock();

cout << "Result time: " << double((end\_time - time)) << endl;

print\_matr(result, N1, M2);

cout << "лента 2 нить" << endl;

time = clock();

result = lent\_mult(matr1, matr2, N1, M1, N2, M2,2);

end\_time = clock();

cout << "Result time: " << double((end\_time-time))<< endl;

print\_matr(result, N1, M2);

cout << "лента 3 нить" << endl;

time = clock();

result = lent\_mult(matr1, matr2, N1, M1, N2, M2,3);

end\_time = clock();

cout << "Result time: " << double((end\_time - time)) << endl;

print\_matr(result, N1, M2);

cout << "лента 4 нить" << endl;

time = clock();

result = lent\_mult(matr1, matr2, N1, M1, N2, M2,4);

end\_time = clock();

cout << "Result time: " << double((end\_time - time)) << endl;

print\_matr(result, N1, M2);

}