Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский институт)»

(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Программное обеспечение ЭВМ и информационные технологии

О т ч ё т

п о л а б о р а т о р н о й р а б о т е 4

Дисциплина: Анализ Алгоритмов

Тема лабораторной работы работы: Параллельные вычисления

Студентки гр. ИУ7-51б _____ **Сушина А.Д.**

Преподаватель _____ **Волкова Л.Л.**

Москва, 2019г

Введение

Параллельные вычислительные системы — это физические компьютерные, а также программные системы, реализующие тем или иным способом параллельную обработку данных на многих вычислительных узлах.

Писать программы для параллельных систем сложнее, чем для последовательных[3], так как конкуренция за ресурсы представляет новый класс потенциальных ошибок в программном обеспечении (багов), среди которых состояние гонки является самой распространённой. Взаимодействие и синхронизация между процессами представляют большой барьер для получения высокой производительности параллельных систем. В последние годы также стали рассматривать вопрос о потреблении электроэнергии параллельными компьютерами. Характер увеличения скорости программы в результате распараллеливания объясняется законами Амдала и Густавсона.

Пото́к выполне́ния (тред; от англ. *Thread* — нить) — наименьшая единица обработки, исполнение которой может быть назначено ядром операционной системы. Реализация потоков выполнения и процессов в разных операционных системах отличается друг от друга, но в большинстве случаев поток выполнения находится внутри процесса. Несколько потоков выполнения могут существовать в рамках одного и того же процесса и совместно использовать ресурсы, такие как память, тогда как процессы не разделяют этих ресурсов. В частности, потоки выполнения разделяют инструкции процесса (его код) и его контекст (значения переменных, которые они имеют в любой момент времени). В качестве аналогии потоки выполнения процесса можно уподобить нескольким вместе работающим поварам. Все они готовят одно блюдо, читают одну и ту же кулинарную книгу с одним и тем же рецептом и следуют его указаниям, причём не обязательно все они читают на одной и той же странице.

Целью данной лабораторной работы является исследование многопоточности на примере алгоритма Винограда.

Задачи лабораторной работы:

- изучить работу многопоточных приложений;
- реализовать алгоритм Винограда с использование многопоточности;
- провести замеры времени многопоточной и однопоточной реализации.

1. Аналитическая часть

1.1. Описание алгоритма

Алгоритм Винограда умножения матриц основан на снижении доли умножений в алгоритме. Предполагается, что некоторые произведения можно вычислить заранее, а затем переиспользовать при вычислении произведений матриц.

Рассмотрим два вектора $V=(v1, v2, v3, v4)$ и $W=(w1, w2, w3, w4)$.

Их скалярное произведение равно:

$$V * W = v1\,w1 + v2\,w2 + v3\,w3 + v4\,w4$$

Это равенство можно переписать в виде:

$$V * W = (v1 + w2)(v2 + w1) + (v3 + w4)(v4 + w3) - v1\,v2 - v3\,v4 - w1\,w2 - w3\,w4.$$

Несмотря на то, что второе выражение требует вычисления большего количества операций, чем первое: вместо четырех умножений - шесть, а вместо трех сложений - десять, выражение в правой части последнего равенства допускает предварительную обработку: его части можно вычислить заранее и запомнить для каждой строки первой матрицы и для каждого столбца второй, что позволяет выполнять для каждого элемента лишь первые два умножения и последующие пять сложений, а также дополнительно два сложения.

Алгоритм Винограда состоит из следующих шагов:

1. Вычисление горизонтальных произведений MulH

$$MulH_i = \sum_{k=0}^{N/2} A_{i,2k} * A_{i,2k+1}$$

2. Вычисление вертикальных произведений MulV

$$MulM_j = \sum_{k=0}^{N/2} B_{2k,j} * B_{2k+1,j}$$

3. Вычисление матрицы результата

$$c_{ij} = -MulH_i - MulV_j + \sum_{k=0}^{N/2} (A_{i,2k} + B_{2k+1,j}) * (A_{i,2k+1} + B_{2k,j})$$

4. Корректирование матрицы в случае нечетного N

$$C_{i,j} = C_{i,j} + A_{i,N-1} * B_{N-1,j}$$

2. Конструкторская часть

2.1. Разработка алгоритмов

На рисунках 1-3 представлена схема алгоритма Винограда.

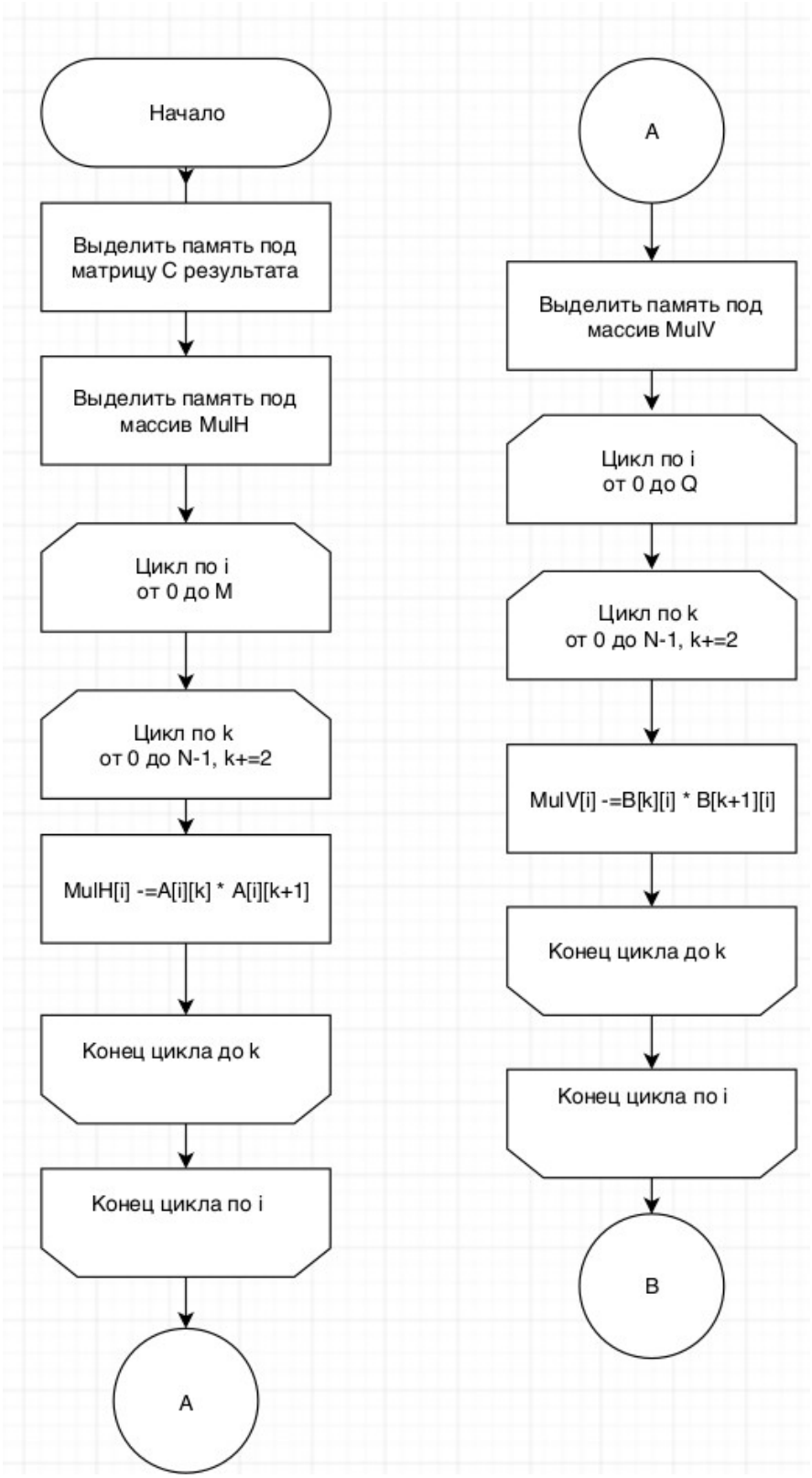

Рис 1. Схема алгоритма Винограда (часть 1)

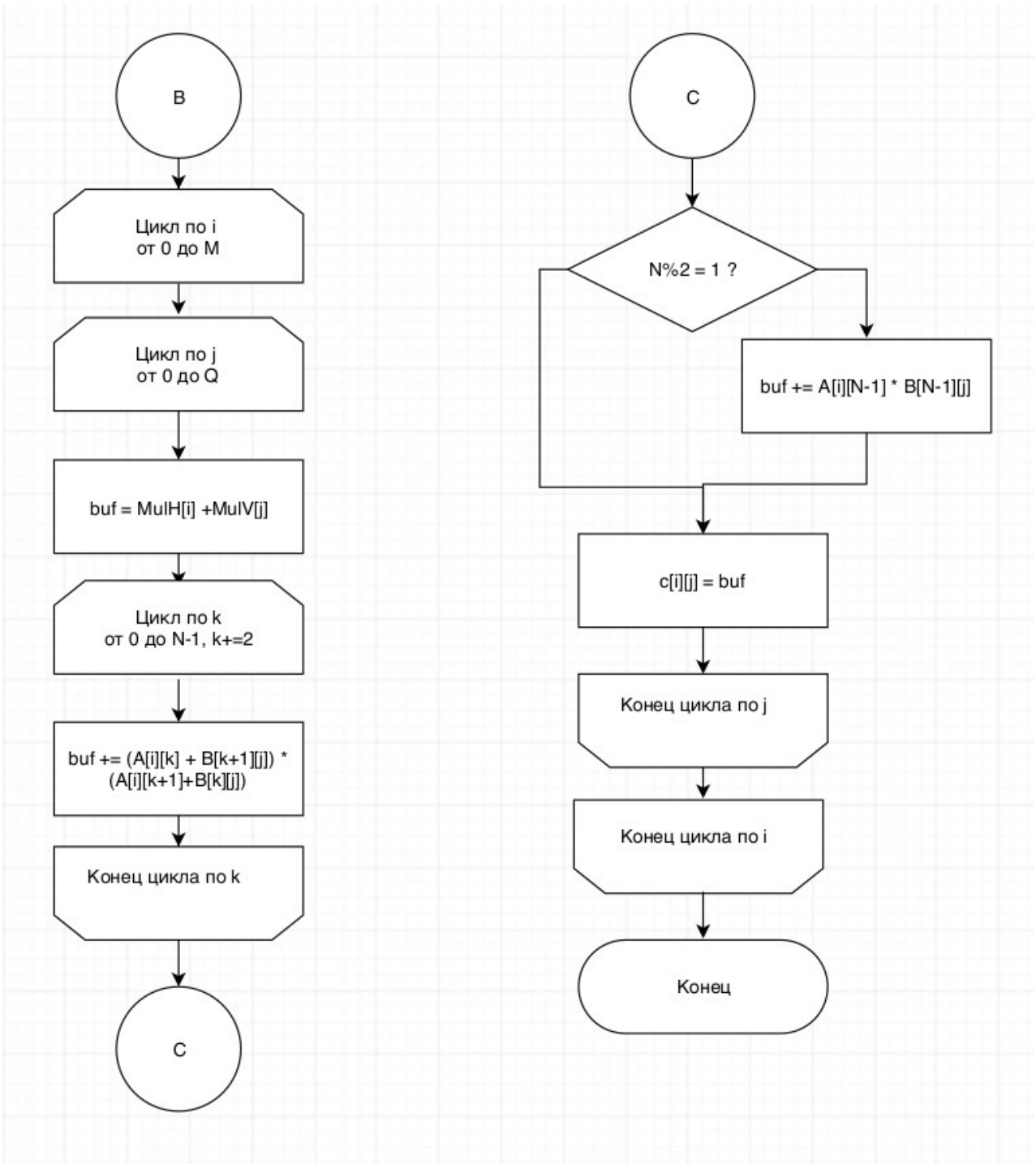

Рис 2. Схема алгоритма Винограда(часть 2)

2.2. Распараллеливание программы

Распараллеливание программы должно увеличить скорость работы. Исходя из схемы, можно разделить программу на логические части, которые можно запускать в параллельных потоках. К примеру, массивы MulH и Mulv можно вычислять параллельно. Также потоки могут считать разные элементы одного массива или одной матрицы, поэтому вычисление MulH и MulV можно разбить еще на несколько потоков, которые будут считать отдельные элементы. Тройной цикл вычисления результата можно распараллелить по строкам. Например, выделить вычисление с 1 по P строку в одном потоке, а с P по M в другом потоке.

В итоге при делении на P потоков:

- Цикл вычисления MulH делим на P/2 потоков: каждый i-ый поток вычисляет элементы массива с i*M/(p/2) по (i+1) *M/(p/2). То есть каждый птоок считает элементы с шагом M/(p/2) до M.
- Цикл вычисления MulV делим на P-P/2 потоков: каждый i-ый поток вычисляет элементы массива с i*Q/(p - p/2) по (i+1) *Q/(p - p/2). То есть каждый птоок считает элементы с шагом Q/(p-p/2) до Q.
- Цикл вычисления матрицы делим на P потоков: каждый i-ый поток вычисляет строки с i*M/p до (i+1)*M/p.

3. Технологическая часть

3.1. Требования к программному обеспечению

На вход программа получает две матрицы с размерами MxN и NxQ. На выходе получается матрица размером MxQ.
Так же должна быть реализована функция для тестирования и функции замера времени.

3.2. Средства реализации

Для реализации программы был выбран язык C++ в связи с возможностью прибегать к использованию ООП, а так же с моими личным опытом работы с этим ЯП. В языке C++ также доступно создание нескольких потоков, что удовлетворяет условиям лабораторной работы. Среда разработки — Qtcreator. Для работы с матрицами были реализован свой класс Matrix.

Замеры времени были проведены с помощью функции clock() из библиотеки ctime.

3.3. Листинг кода

На листингах 1-5 представлен код полученной программы.

Листинг 1. Реализация оптимизированного алгоритма Винограда

```
Matrix Vinograd(Matrix A, Matrix B) {
    int N = A.cols();
    int M = A.rows();
    int Q = B.cols();
    Matrix c(M, Q);

    std::vector<int> MulH(M,0);
    for (int i = 0; i < M; i++) {
        for (int k = 0; k < N-1; k += 2) {
            MulH[i] -= A[i][k] * A[i][k+1];
        }
    }

    std::vector<int> MulV(Q,0);
    for (int i = 0; i < Q; i++) {
        for (int k = 0; k < N-1; k += 2) {
            MulV[i] -= B[k][i]*B[k+1][i];
        }
    }

    for (int i = 0; i < M; i++) {
        for (int j = 0; j < Q; j++) {
            int buf = MulH[i] + MulV[j];
            for (int k = 0; k < N-1; k += 2) {
                buf += (A[i][k] + B[k+1][j])*(A[i][k+1] + B[k][j]);
            }
            if (N%2 == 1) {
                 buf += A[i][N-1]*B[N-1][j];
            }
            c[i][j] = buf;
        }
    }
```

```
    return c;
```

Листинг 2. Первый шаг алгоритма Винограда

```
void first (std::vector<int> &MulH, Matrix A, int N, int start, int end)
{
    for (int i = start; i < end; i++) {
        for (int k = 0; k < N-1; k += 2) {
            MulH[i] -= A[i][k] * A[i][k+1];
        }
    }
}
```

Листинг 3. Второй шаг алгоритма Винограда

```
void second (std::vector<int> &MulV, Matrix B, int N, int start, int end) {
    for (int i = start; i < end; i++) {
        for (int k = 0; k < N-1; k += 2) {
            MulV[i] -= B[k][i]*B[k+1][i];
        }
    }
}
```

Листинг 4. Третий шаг алгоритма Винограда

```
void third (Matrix &c, Matrix A, Matrix B, int N, std::vector<int> MulH,
std::vector<int> MulV, int starti, int endi, int startj, int endj){
    for (int i = starti; i < endi; i++) {
        for (int j = startj; j < endj; j++) {
            int buf = MulH[i] + MulV[j];
            for (int k = 0; k < N-1; k += 2) {
                buf += (A[i][k] + B[k+1][j])*(A[i][k+1] + B[k][j]);
            }
            if (N%2 == 1) {
                 buf += A[i][N-1]*B[N-1][j];
            }
            c[i][j] = buf;
        }
    }
}
```

Листинг 5. Функция распараллеливания алгоритма Винограда

```
Matrix Mul(Matrix A, Matrix B, int tcount) {
    int N = A.cols();
    int M = A.rows();
    int Q = B.cols();
    Matrix c(M, Q);

    std::vector<int> MulH(M,0);
    std::vector<int> MulV(Q,0);

    thread *th[tcount];
    int n = tcount/2;
    for (int i = 0; i < n; i++) {
        th[i] = new thread(first, ref(MulH), A, N, i*M/n, (i+1)*M/n);
    }
    int m = tcount - n;
    for (int i = 0; i < m; i++) {
        th[n+i] = new thread(first, ref(MulV), B, N, i*M/m, (i+1)*M/m);
    }

    for (int i = 0; i < tcount; i++) {
        th[i]->join();
    }
```

```
    thread *th3[tcount];
    for (int i = 0; i < tcount; i++) {
         th3[i] = new thread(third,
                        ref(c),
                        A, B, N,
                        MulH, MulV,
                        i*M/tcount, (i+1)*M/tcount,
                        0, Q);
    }
    for (int i = 0; i < tcount; i++) {
        th3[i]->join();
    }
    for (int i = 0; i < tcount; i++) {
        delete th[i];
        delete th3[i];
    }
    return c;
}
```

4.Экспериментальная часть

4.1. Постановка эксперимента по замеру времени

Были проведены временные эксперименты для матриц от 100x100 до 1000x1000 с шагом 100 и для матриц от 101x101 до 1001x1001 с шагом 100. Для каждого замера взят средний результат из 5 замеров. Результаты замеров представлены на рисунке 3.

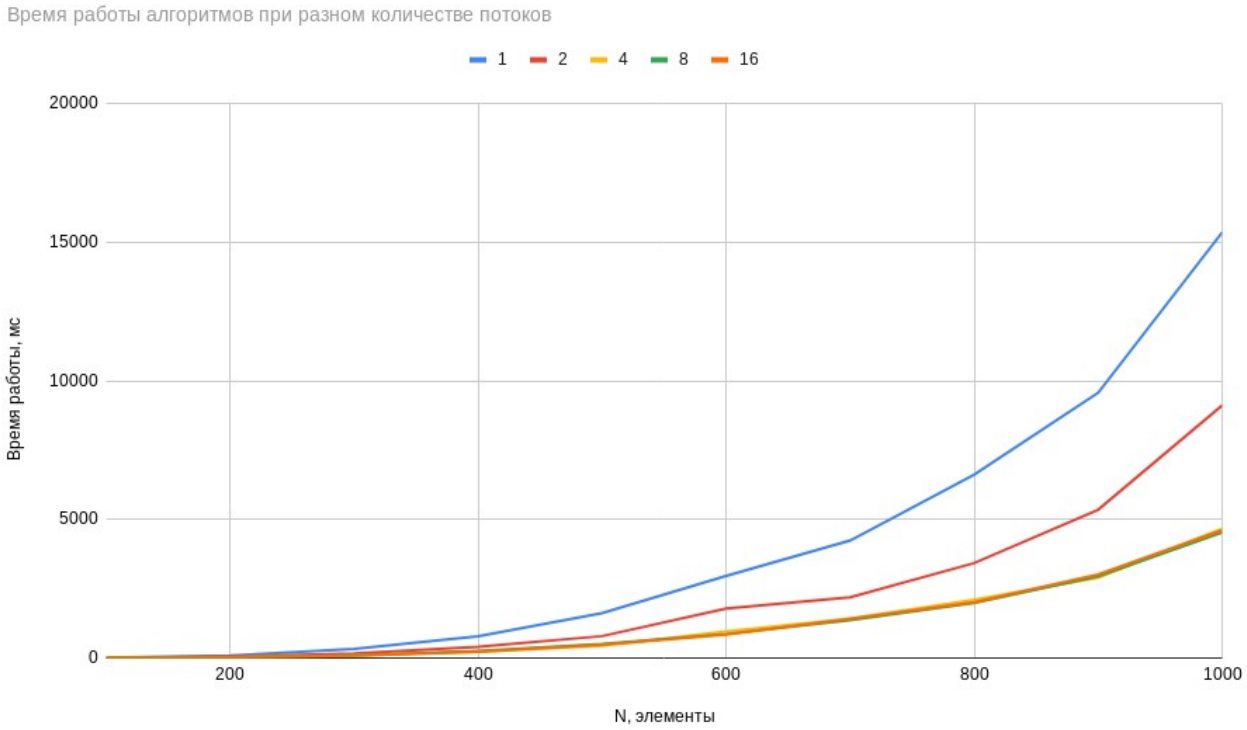

Рис 3. Диаграмма результатов замера времени

Заключение

В ходе лабораторной работы была написана многопоточная реализация алгоритма Винограда. Были проведены замеры времени работы алгоритма при разном количестве потоков. Алгоритм на 2 потоках работает почти в 2 раза быстрее однопоточного алгоритма., а алгоритм на 4х потоках работает в 4 раза быстрее. Реализации с большим количеством потоков также дают выигрыш во времени, однако он незначителен.