Государственное образовательное учреждение высшего
профессионального образования
«Московский государственный технический
университет имени Н. Э. Баумана»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

ЛАБОРАТОРНАЯ РАБОТА №2

на тему:

«Распараллеливание алгоритмов умножения матриц»

Студент ИУ7-54: Морозов И. А
Преподаватель: Погорелов. Д. А.

Москва 2018

Введение

Целью данной лабораторной работы является изучение и сравнение двух подходов к реализации алгоритмов умножения матриц(классический алгоритм умножения матриц и алгоритм Винограда) : с использование параллельных вычислений в нескольких потоках и с использованием одного потока.

1. Аналитическая часть

1.1. Описание алгоритмов

1.1.1. Алгоритм перемножения матриц

Описание алгоритма:

Пусть даны две прямоугольные матрицы A и B размерности $l \times m$ и $m \times n$ соответственно:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{l1} & a_{l2} & \cdots & a_{lm} \end{bmatrix}, \quad B = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mn} \end{bmatrix}.$$

Тогда матрица C размерностью $l \times n$:

$$C = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{l1} & c_{l2} & \cdots & c_{ln} \end{bmatrix},$$

В которой:

$$c_{ij} = \sum_{r=1}^{m} a_{ir} b_{rj} \quad (i = 1, 2, \ldots l;\ j = 1, 2, \ldots n)\,.$$

Называется их произведением

1.1.2. Алгоритм Винограда

Описание алгоритма:

Усовершенствованный алгоритм умножения матриц таким образом, что если посмотреть на результат умножения двух матриц, то видно, что каждый элемент в нем представляет собой произведение соответствующих строки и столбца исходных матриц. Можно заметить также, что такое умножение допускает предварительную обработку, позволяющую часть работы выполнить заранее. Рассмотрим два вектора:

$$V = (v1,v2,v3,v4) \text{ и } W = (w1,w2,w3,w4)$$

Их скалярное произведение равно:

$$V \cdot W = v1w1 + v2w2 + v3w3 + v4w4$$

Это равенство можно переписать в виде:

$$V \cdot W = (v1 + w2)(v2 + w1) + (v3 + w1)(v4 + w3) - v1v2 - v3v4 - w1w2 - w3w4$$

1.2. Параллельные вычисления

Множественные нити исполнения в одном процессе называют потоками и это базовая единица загрузки ЦПУ, состоящая из идентификатора потока, счетчика, регистров и стека. Потоки внутри одного процесса делят секции кода, данных, а также различные ресурсы: описатели открытых файлов, учетные данные процесса сигналы, значения umask, nice, таймеры и прочее.

У всех исполняемых процессов есть как минимум один поток исполнения. Некоторые процессы этим и ограничиваются в тех случаях, когда дополнительные нити исполнения не дают прироста производительности, но только усложняют программу. Однако таких программ с каждым днем становится относительно меньше.

Существует закономерность между количеством параллельных нитей исполнения процесса, алгоритмом программы и ростом производительности. Это зависимость называется «Законом Амдаля»:

$$Acceleration = \frac{1}{F + \frac{(1-F)}{N}}$$

Используя уравнение, показанное на рисунке, можно вычислить максимальное улучшение производительности системы, использующей N процессоров и фактор F, который указывает, какая часть системы **не** может быть распараллелена. Например 75% кода запускается параллельно, а 25% — последовательно. В таком случае на двухядерном процессоре будет достигнуто 1.6 кратное ускорение программы, на четырехядерном процессоре — 2.28571 кратное, а предельное значение ускорения при N стремящемся к бесконечности равно 4.

2. Конструкторская часть

Для исследований будут реализованы: классический алгоритм умножения матриц, алгоритм Винограда, далее будет произведено распараллеливание данных алгоритмов.

2.1 Разработка алгоритмов

2.1.1. Схема алгоритма перемножения матриц

Входные данные:

matrix1 - первая матрица
m - количество строк первой матрицы
n1 - количество столбцолв первой матрицы
Matrix2 - вторая матрица
n2 - количество срок второй матрицы
q - количество столбцов второй матрицы

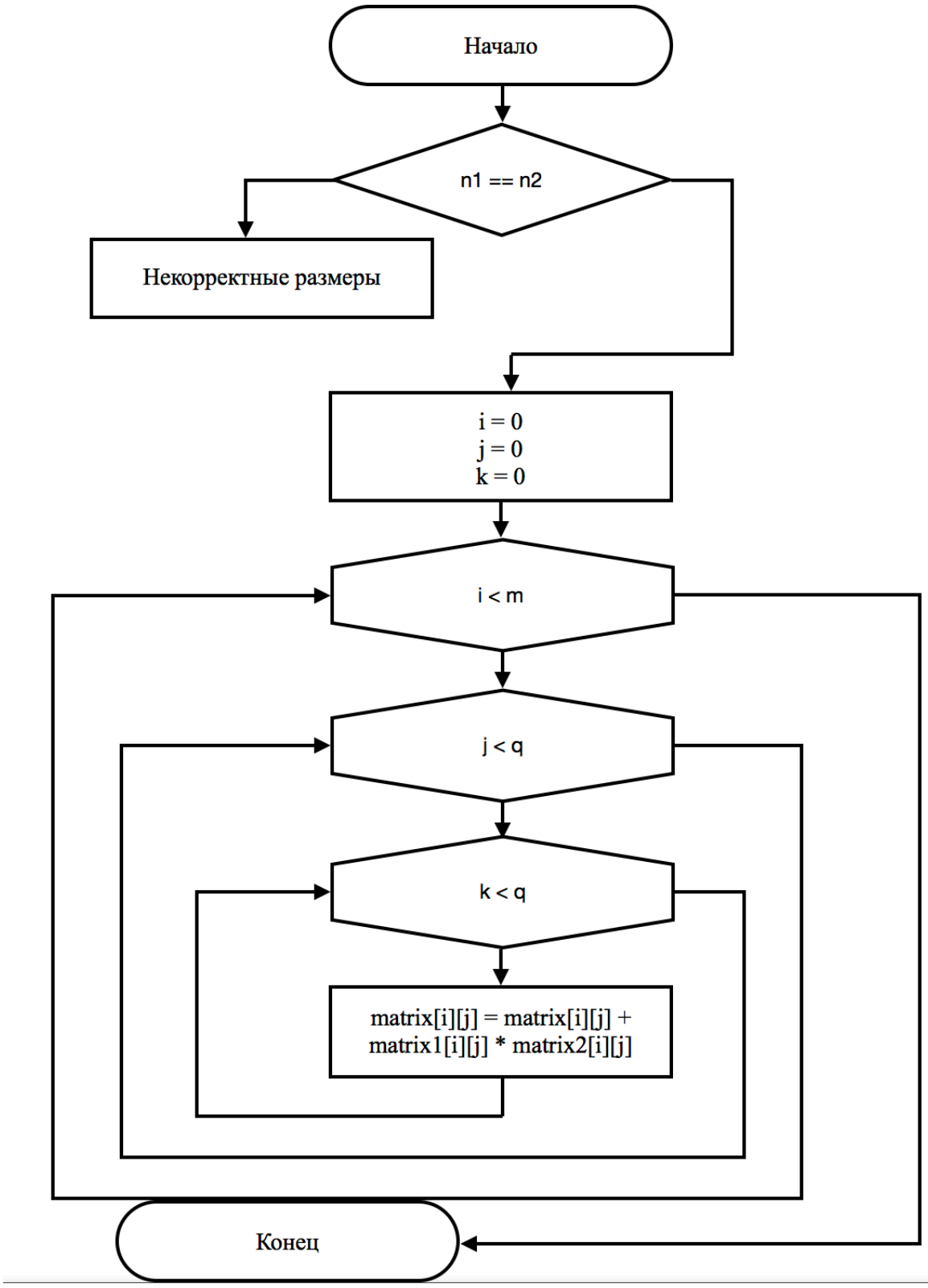


Рисунок 1. Схема алгоритма классического умножения матриц.

2.1.2. Схема алгоритма Винограда

Входные данные:

matrix1 - первая матрица

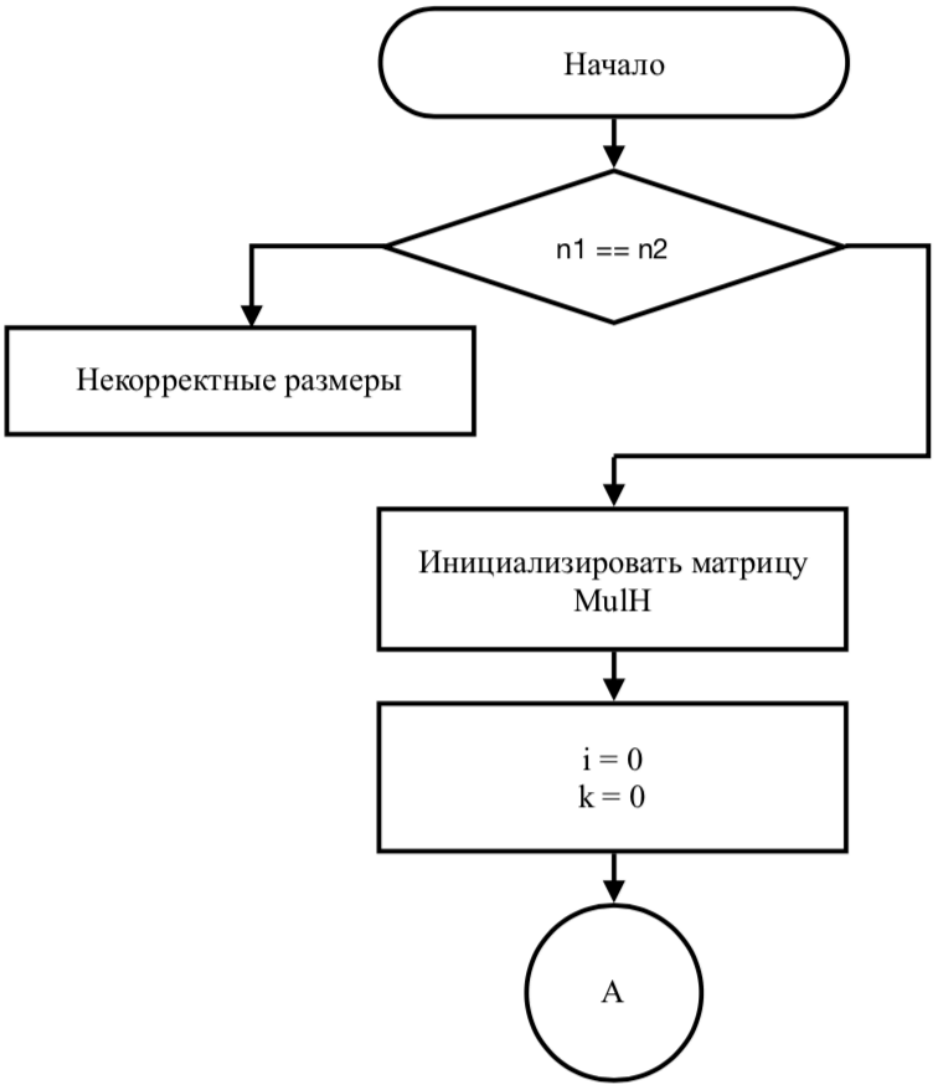
m - количество строк первой матрицы

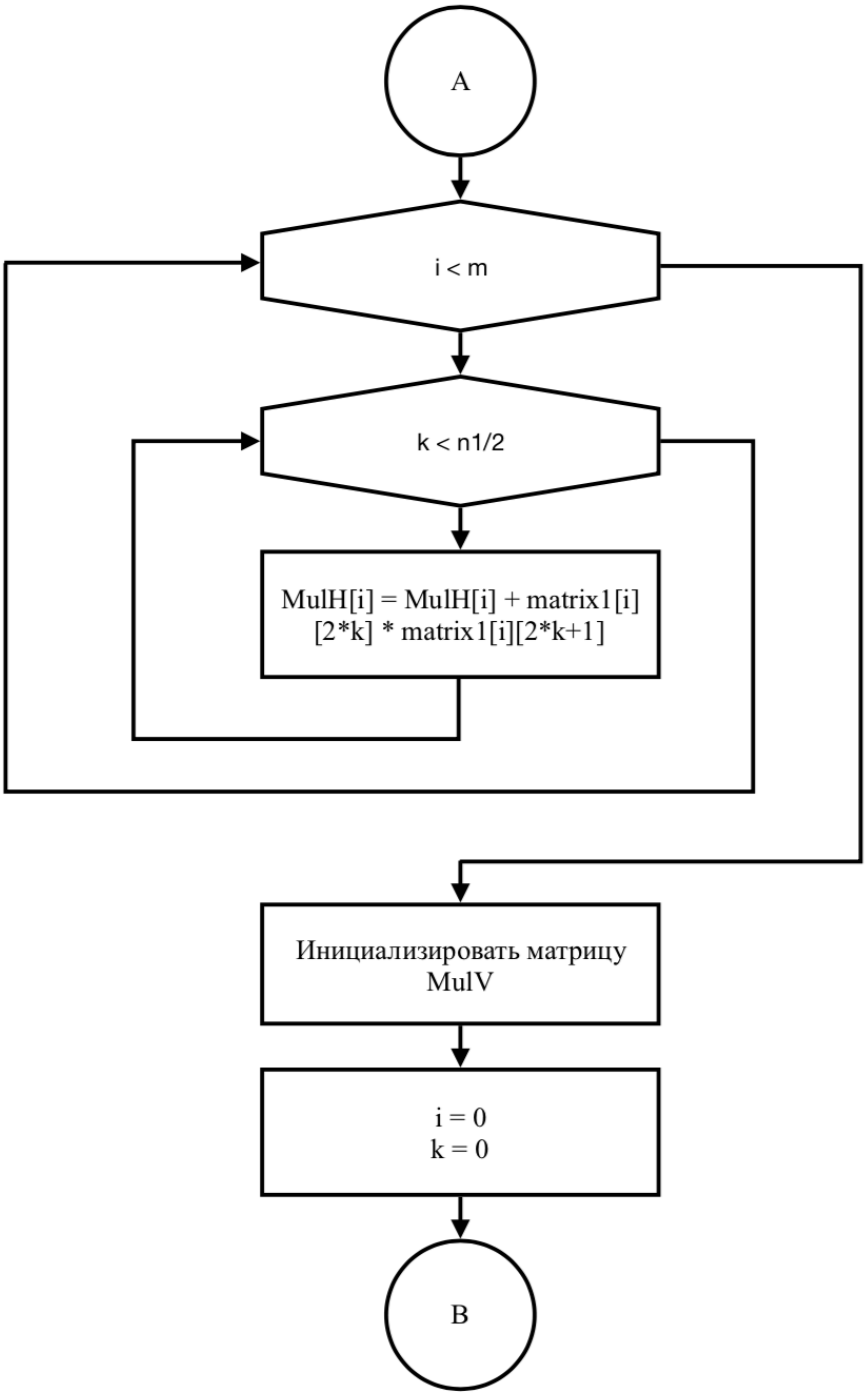
n1 - количество столбцолв первой матрицы

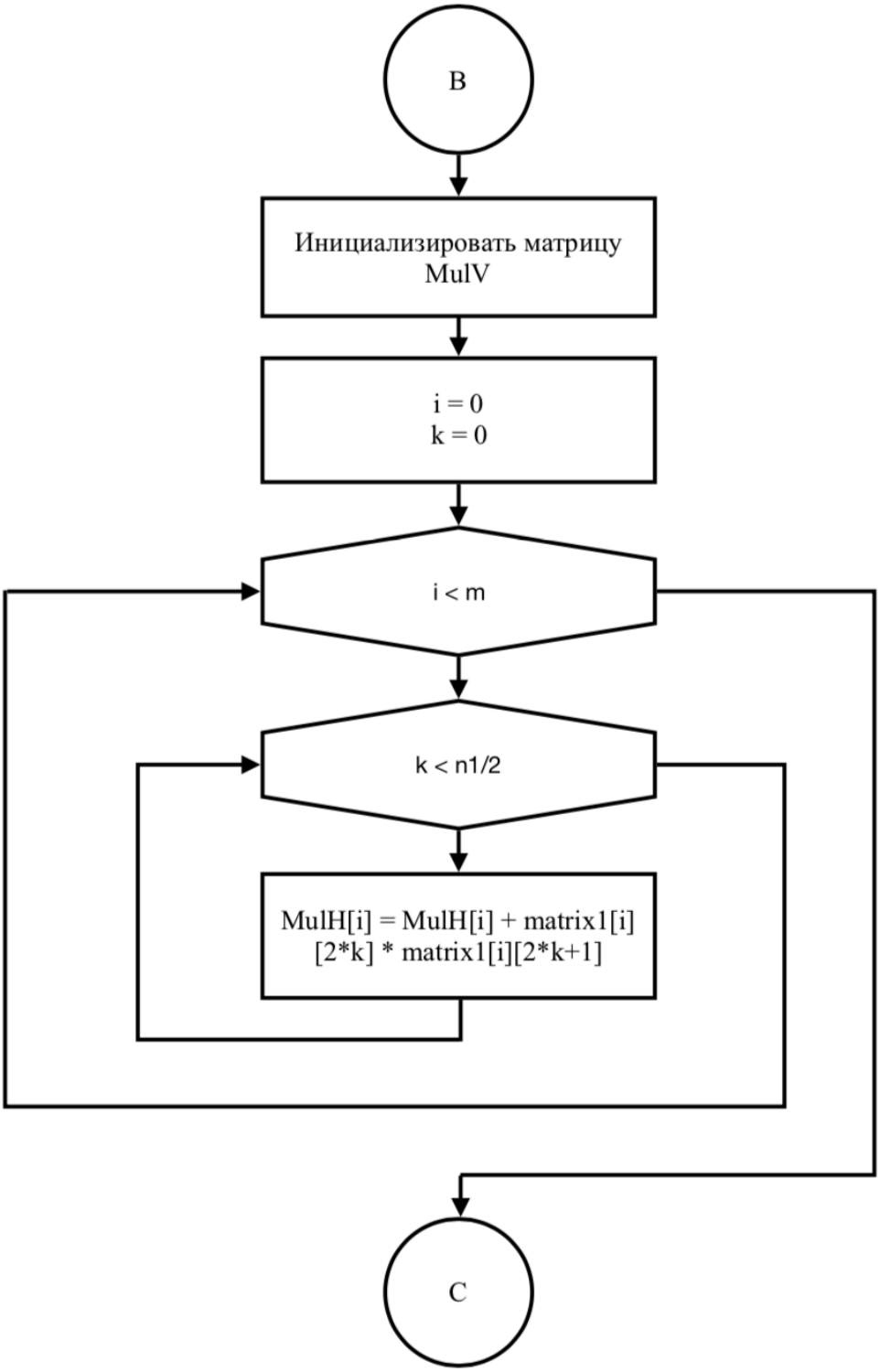
Matrix2 - вторая матрица

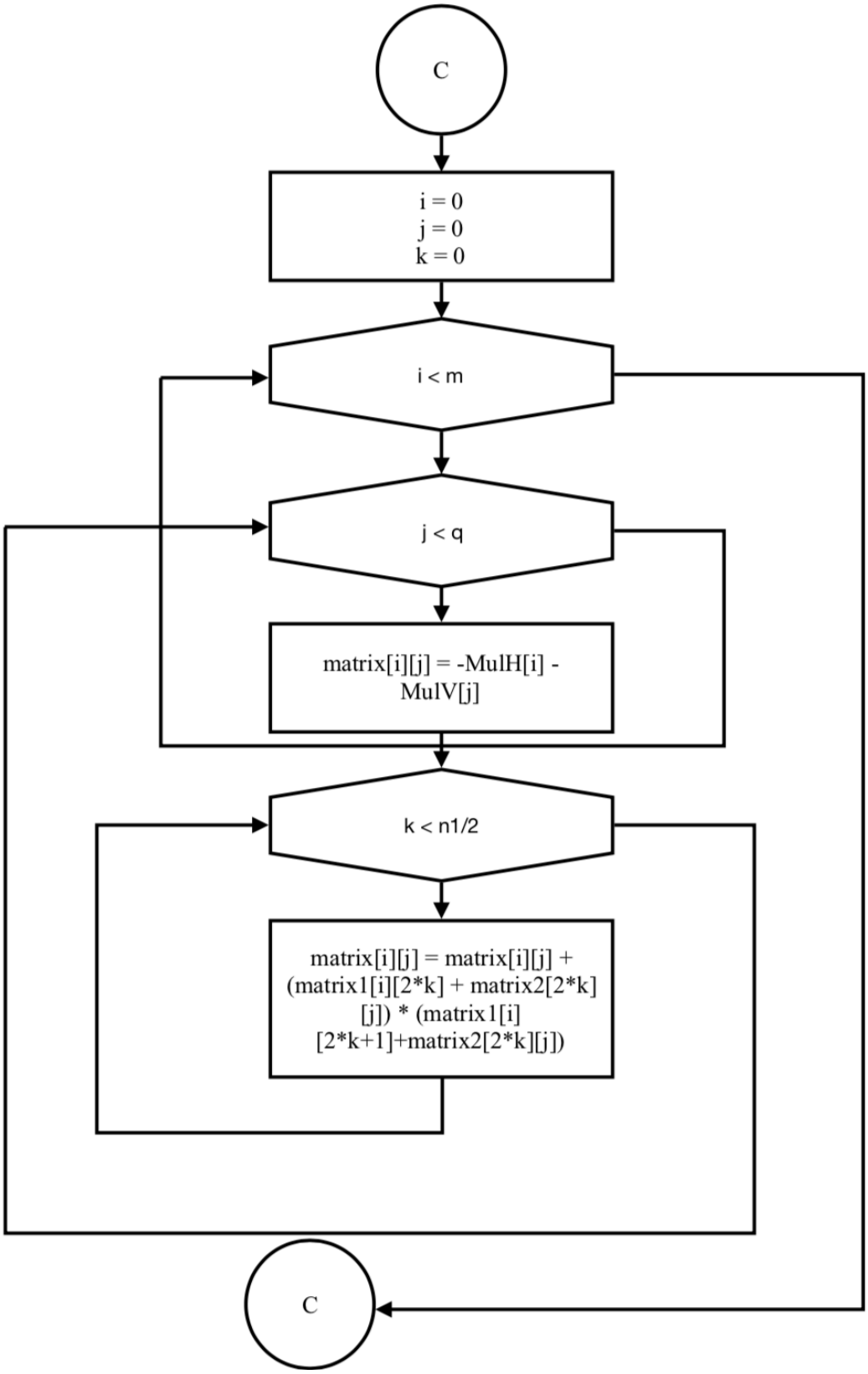
n2 - количество срок второй матрицы

q - количество столбцов второй матрицы




A
i < m
k < n1/2
MulH[i] = MulH[i] + matrix1[i]
[2*k] * matrix1[i][2*k+1]
Инициализировать матрицу
MulV
i = 0
k = 0
B


B
Инициализировать матрицу MulV
i = 0
k = 0
i < m
k < n1/2
MulH[i] = MulH[i] + matrix1[i][2*k] * matrix1[i][2*k+1]
C


C
i = 0
j = 0
k = 0
i < m
j < q
matrix[i][j] = -MulH[i] -
MulV[j]
k < n1/2
matrix[i][j] = matrix[i][j] +
(matrix1[i][2*k] + matrix2[2*k]
[j]) * (matrix1[i]
[2*k+1]+matrix2[2*k][j])
C

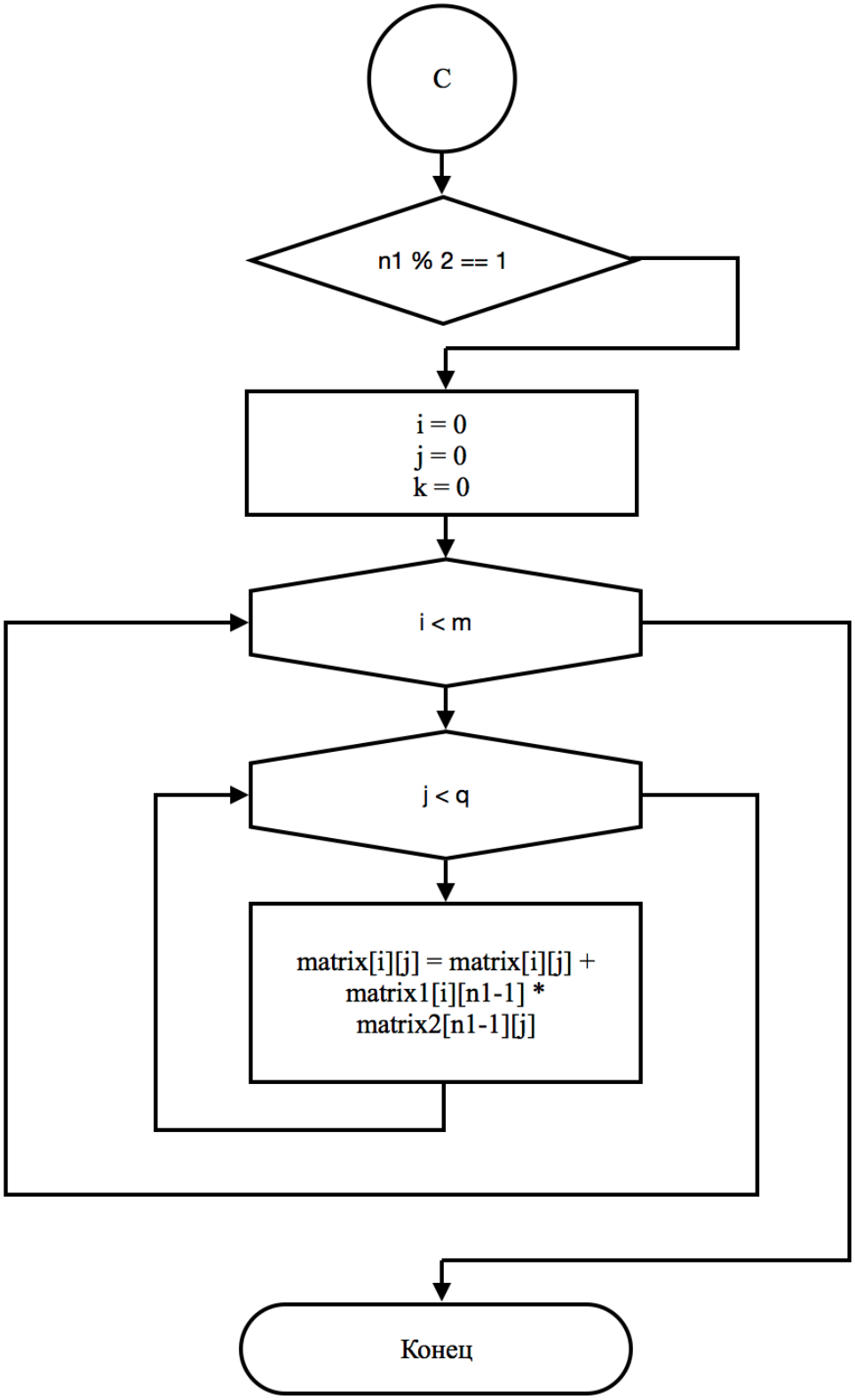
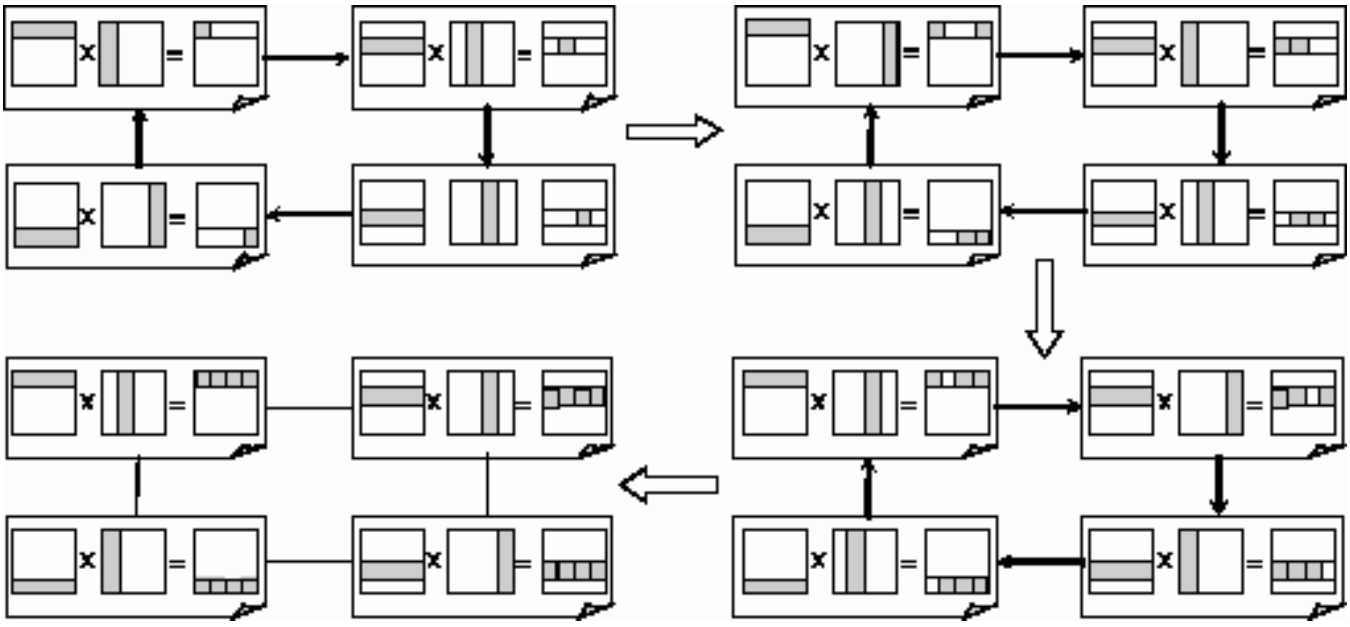


Рисунок 2. Схема алгоритма Винограда

2.1.3. Распаралеленный классический алгоритм умножения матриц

Предлагается разбить первую матрицу на две части по строкам и вторую матрицу по столбцам, далее в одном потоке выполнить умножение первой половины, а во втором второй половины матрицы. В данной ситуации можно распараллелить около 20% алгоритма на двух потоках, следовательно по закону Амдаля прирост должен составить 1.5 раза.



2.1.4. Распараллеленный алгоритм Винограда

Предлагается вычисление сумм столбцов и строк выполнять в отдельных потоках и так же как и в распараллеливании классического алгоритма поделить матрицы на две части. В данной ситуации нельзя распараллелить около 30% кода, значит по закону Амдаля на двух процессах прирост составит ~1.33

3. Технологическая часть

В данном разделе будет представленно описание используемого языка программирования, а так же будет показан листинг кода функций, работающих согласно указанным выше алгоритмам.

3.1. Требования к программному обеспечению

Данная программа была реализована на языке C++ компилятор для которого поддерживается многими операционными системами. Для управлениями потоками используется встроенная библиотека pthread

Компилятор: g++

3.2. Средства реализации

Программа была реализована на операционной система MacOS в среде разработки Xcode

3.3 Листинги кода

Здесь я приведу листинги кода параллельных программ, так как листинги кода однопоточных программ были рассмотрены в лабораторной работе №2.

3.3.1. Листинг распаралеленного алгоритма умножения матриц

mult - структура содержащая размеры введенных матриц и матрицы для вычисления
thread - массив потоков

```
void *first_half(void *args)
{
    mult_matrix *mult = (mult_matrix*) args;
```

```
    int half_m = (mult->M)/2;

    for( int i = 0; i < half_m; ++i )
    {
        for( int j = 0; j < mult->Q; ++j )
        {
            for( int k = 0; k < mult->N1; ++k )
            {
                mult->result[i][j] = mult->result[i][j] +
mult->matrix1[i][k] * mult->matrix2[k][j];
            }
        }
    }
    return args;
}

void *second_half(void *args)
{
    mult_matrix *mult = (mult_matrix*) args;
    int half_m = (mult->M)/2;

    for( int i = half_m; i < mult->M; ++i )
    {
        for( int j = 0; j < mult->Q; ++j )
        {
            for( int k = 0; k < mult->N1; ++k )
            {
                mult->result[i][j] = mult->result[i][j] +
mult->matrix1[i][k] * mult->matrix2[k][j];
            }
        }
    }
    return args;
}

int multiply_matrix(mult_matrix *mult)
{
    pthread_t thread[2];
    int errflag;

    errflag = pthread_create(&thread[0], NULL,
first_half, &mult);
    if(errflag != 0) std::cout << "First thread drop" <<
std::endl;
    errflag = pthread_create(&thread[1], NULL,
second_half, &mult);
    if(errflag != 0) std::cout << "Second thread drop" <<
std::endl;

    errflag = pthread_join(thread[0], NULL);
    if(errflag != 0) std::cout << "First thread can't
stop" << std::endl;
```

```
    errflag = pthread_join(thread[1], NULL);
    if(errflag != 0) std::cout << "Second thread can't
stop" << std::endl;
    return errflag;
}
```

3.3.2 Листинг распаралеленного алгоритма Винограда

mult - структура содержащая размеры введенных матриц и матрицы для вычисления

thread - массив потоков

```
void* rows(void *args)
{
    vinograd *mult = (vinograd *) args;
    mult->MulH = std::vector<int> (mult->M, 0);

    for( int i = 0; i < mult->M; ++i )
    {
        for( int k = 0; k < mult->half_n; ++k )
        {
            mult->MulH[i] += mult->matrix1[i][(k<<1)] *
mult->matrix1[i][(k<<1)+1];
        }
    }

    return args;
}

void* cols(void *args)
{
    vinograd *mult = (vinograd *) args;
    mult->MulV = std::vector<int> (mult->Q, 0);

    for( int i = 0; i < mult->Q; ++i )
    {
        for( int k = 0; k < mult->half_n; ++k )
        {
            mult->MulV[i] += mult->matrix2[k<<1][i] *
mult->matrix2[(k<<1)+1][i];
        }
    }
```

```
    return args;
}

void *first_half_res(void *args)
{
    vinograd *mult = (vinograd *) args;
    int half_m = mult->M/2;

    for( int i = 0; i < half_m; ++i )
    {
        for( int j = 0; j < mult->Q; ++j )
        {
            mult->result[i][j] = -mult->MulH[i] - mult-
>MulV[j];
            for( int k = 0; k < mult->half_n; ++k )
            {
                mult->result[i][j] += ( mult->matrix1[i]
[k<<1] + mult->matrix2[(k<<1)+1][j] ) *
                ( mult->matrix1[i][(k<<1)+1] + mult-
>matrix2[k<<1][j] );
            }
        }
    }

    return args;
}

void *second_half_res(void *args)
{
    vinograd *mult = (vinograd *) args;
    int half_m = mult->M/2;

    for( int i = half_m; i < mult->M; ++i )
    {
        for( int j = 0; j < mult->Q; ++j )
        {
            mult->result[i][j] = -mult->MulH[i] - mult-
>MulV[j];
            for( int k = 0; k < mult->half_n; ++k )
            {
                mult->result[i][j] += ( mult->matrix1[i]
[k<<1] + mult->matrix2[(k<<1)+1][j] ) *
                ( mult->matrix1[i][(k<<1)+1] + mult-
>matrix2[k<<1][j] );
            }
        }
    }

    return args;
}

void *first_half(void *args)
```

```
{
    vinograd *mult = (vinograd *) args;
    int half_m = mult->M/2;

    for( int i = 0; i < half_m; ++i )
    {
        for( int j = 0; j < mult->Q; ++j )
        {
            mult->result[i][j] += mult->matrix1[i][mult-
>N1 - 1] * mult->matrix2[mult->N1 - 1][j];
        }
    }

    return args;
}

void *second_half(void *args)
{
    vinograd *mult = (vinograd *) args;
    int half_m = mult->M/2;

    for( int i = half_m; i < mult->M; ++i )
    {
        for( int j = 0; j < mult->Q; ++j )
        {
            mult->result[i][j] += mult->matrix1[i][mult-
>N1-1] * mult->matrix2[mult->N1-1][j];
        }
    }

    return args;
}

int vinograd_mult(vinograd *mult)
{
    // Объявляем потоки
    pthread_t thread[NumThreads];
    int errflag;

    // Распараллеливание вычисления сумм колонок и
столбцов
    errflag = pthread_create(&thread[0], NULL, rows,
mult);
    if(errflag != 0) std::cout << "First thread drop" <<
std::endl;
    errflag = pthread_create(&thread[1], NULL, cols,
mult);
    if(errflag != 0) std::cout << "Second thread drop" <<
std::endl;

    // "Сливание" двух созданных потоков с потоком main
    errflag = pthread_join(thread[0], NULL);
```

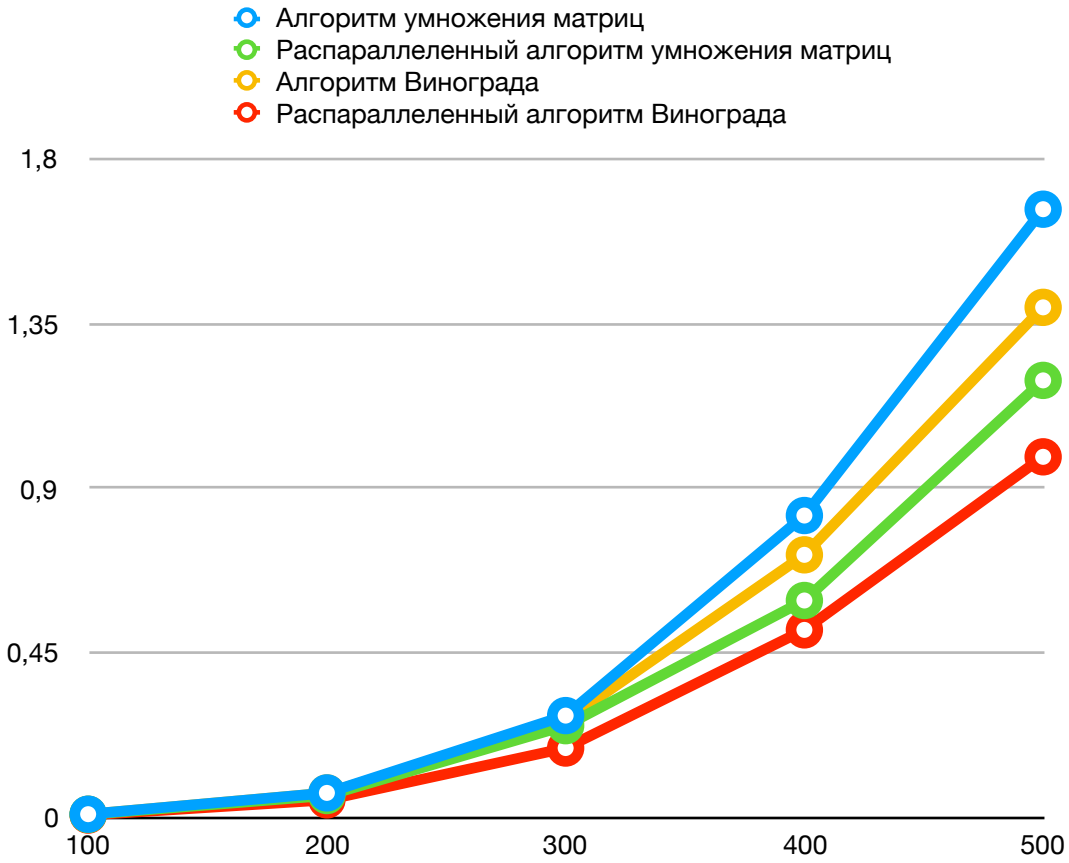
```
    if(errflag != 0) std::cout << "First thread can't
stop" << std::endl;
    errflag = pthread_join(thread[1], NULL);
    if(errflag != 0) std::cout << "Second thread can't
stop" << std::endl;

    // Распараллеливание вычисления результата
    errflag = pthread_create(&thread[2], NULL,
first_half_res, mult);
    if(errflag != 0) std::cout << "Third thread drop" <<
std::endl;
    errflag = pthread_create(&thread[3], NULL,
second_half_res, mult);
    if(errflag != 0) std::cout << "Fourth thread drop" <<
std::endl;

    // "Сливание" двух созданных потоков с потоком main
    errflag = pthread_join(thread[2], NULL);
    if(errflag != 0) std::cout << "Third thread can't
stop" << std::endl;
    errflag = pthread_join(thread[3], NULL);
    if(errflag != 0) std::cout << "Fourth thread can't
stop" << std::endl;

    // Если размеры были нечетные
    if( mult->N1 % 2 == 1 )
    {
        // Распараллеливание вычислений результата при
нечетных размерах
        errflag = pthread_create(&thread[4], NULL,
first_half, mult);
        if(errflag != 0) std::cout << "Third thread drop"
<< std::endl;
        errflag = pthread_create(&thread[5], NULL,
second_half, mult);
        if(errflag != 0) std::cout << "Fourth thread
drop" << std::endl;

        // "Сливание" двух созданных потоков с потоком
main
        errflag = pthread_join(thread[4], NULL);
        if(errflag != 0) std::cout << "Third thread can't
stop" << std::endl;
        errflag = pthread_join(thread[5], NULL);
        if(errflag != 0) std::cout << "Fourth thread
can't stop" << std::endl;
    }

    return errflag;
}
```

4. Экспериментальная часть

В данном разделе будет представлено сравнение временных характеристик алгоритмов.

4.1. Постановка эксперимента

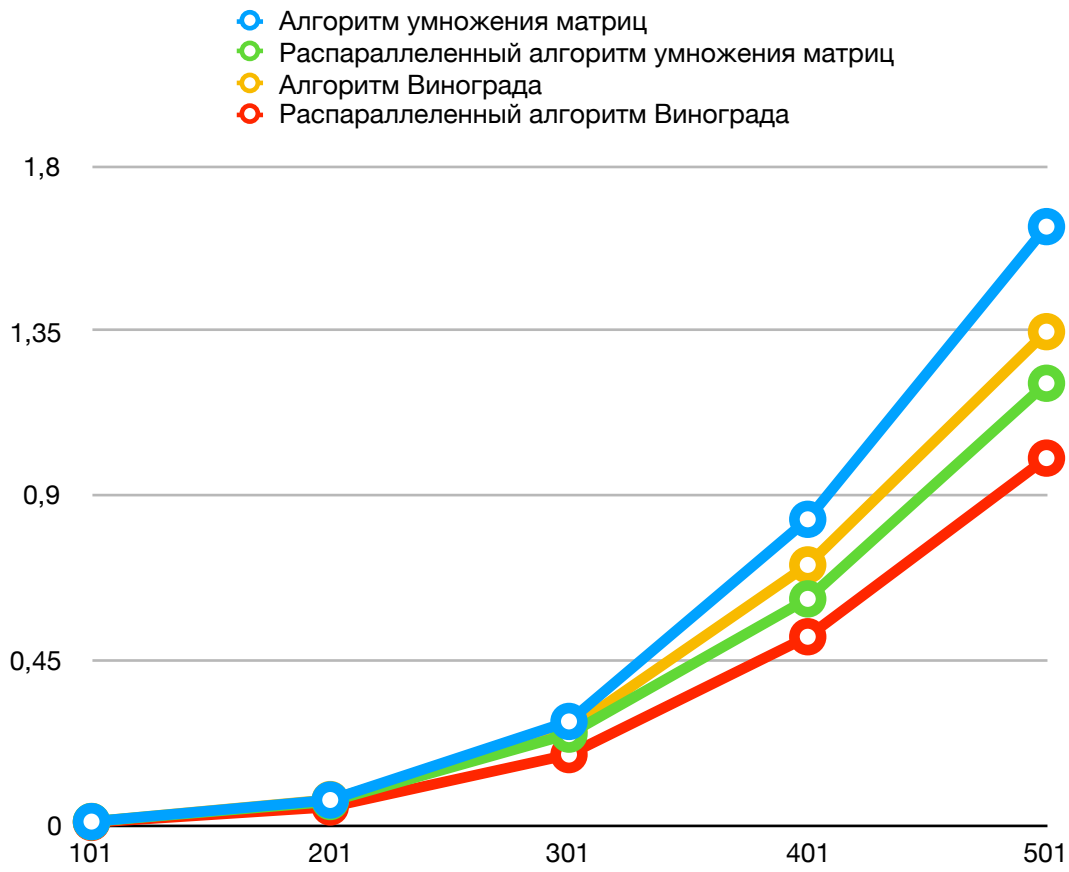
Первый эксперимент производится для лучшего случая на



матрицах с размерами от 100×100 до 500×500 с шагом 100 так как уже на размере 500 для однопоточных алгоритмов приходилось ожидать около минуты. Каждый эксперимент тем не менее был повторен 100 раз для усреднения значений.

Теоретические предположения подтвердились, в случае классического умножения прирост составляет ~1.15, в случае алгоритма Винограда, прирост составит 1.33.

Второй эксперимент производится для худшего случая, когда поданы матрицы с нечетными размерами от 101×101 до 501×501 с шагом 100.



Заключение

В ходе лабораторной работы были изучены и реализованы распараллеленные алгоритмы умножения матриц, а именно: распараллелен классический алгоритм умножения матриц, распараллелен алгоритм Винограда.

Был проведен анализ данных алгоритмов. Распараллеленный алгоритм Винограда показал себя как наиболее эффективный для большинства случаев, что сходится с ожидаемым результатом