



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

ДИСЦИПЛИНА «Анализ алгоритмов»

Лабораторная работа № 4

Тема «Параллельное умножение матриц»

Студент Чалый А.А.

Группа ИУ7 – 52Б

Оценка (баллы) _____

Преподаватели Волкова Л.Л.
Строганов Ю.В.

Москва.
2020 г.

Оглавление

Введение.....	4
1. Аналитическая часть.....	5
1.1 Описание алгоритма умножения матриц.....	5
1.2 Алгоритм Винограда.....	5
1.3 Параллельный алгоритм Винограда.....	6
1.4 Параллельное программирование.....	6
1.4 Организация взаимодействия параллельных потоков.....	7
Вывод.....	7
2. Конструкторская часть.....	8
2.1 Разработка алгоритмов.....	8
2.2 Оптимизации алгоритма Винограда.....	9
2.3 Распараллеливание алгоритма Винограда	9
2.3 Распараллеливание по группам строк.....	9
2.4 Распараллеливание по строке	10
Вывод.....	11
3. Технологическая часть.....	12
3.1 Требования к программному обеспечению.....	12
3.2 Средства реализации.....	12
3.4 Тестирование	19
Вывод.....	20
4. Экспериментальная часть.....	21
4.1 Постановка эксперимента.....	21
4.2 Сравнительный анализ на материале экспериментальных данных	21
Вывод.....	22
Заключение	23
Список использованных источников	24

Введение

Целью данной работы является изучение возможности параллельных вычислений, получение практических навыков реализации алгоритма умножения матриц Винограда с использованием параллельных вычислений, приобретение навыков сравнения зависимости времени работы алгоритма от числа параллельных потоков и размера матриц, а также сравнения стандартного и параллельного алгоритмов.

Требуется реализовать описанные ниже алгоритмы:

- 1) улучшенный алгоритм Винограда;
- 2) параллельный улучшенный алгоритм Винограда.

1. Аналитическая часть

В данном разделе будет рассмотрено описание алгоритма умножения матриц Винограда, а также будет рассказано про параллельное программирование.

1.1 Описание алгоритма умножения матриц

Умножение матриц – одна из основных операций над матрицами. Матрица, получаемая в результате операции умножения, называется произведением матриц. Операция умножения двух матриц выполнима только в том случае, если число столбцов в первом сомножителе равно числу строк во втором: в этом случае говорят, что матрицы согласованы. В частности, умножение всегда выполнимо если оба сомножителя – квадратные матрицы одного и того же порядка. Таким образом, из существования произведения AB вовсе не следует существование произведения BA [1].

Эти алгоритмы активно применяются во всех областях, применяющих линейную алгебру, такие как:

- компьютерная графика;
- физика;
- экономика;
- и так далее.

1.2 Алгоритм Винограда

Рассмотрим два вектора $V = (v_1, v_2, v_3, v_4)$ и $W = (w_1, w_2, w_3, w_4)$. Их скалярное произведение равно:

$$V \cdot W = v_1w_1 + v_2w_2 + v_3w_3 + v_4w_4 \quad (1)$$

Это равенство можно переписать в виде:

$$V \cdot W = (v_1 + w_2)(v_2 + w_1) + (v_3 + w_4)(v_4 + w_3) - v_1v_2 - v_3v_4 - w_1w_2 - w_3w_4 \quad (2)$$

Может показаться, что выражение (2) задает больше работы, чем (1): вместо четырех умножений совершается шесть, а также сложений, вместо трех – десять. Менее очевидно, что 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. Это означает, что над предварительно обработанными элементами нам придется выполнять лишь первые два умножения и последующие пять сложений, а также дополнительные два

сложения. Важным моментом данного алгоритма считается то, что при нечетном количестве столбцов первой матрицы требуется к каждому элементу результирующей матрицы добавить результат произведения последнего столбца первой матрицы на последнюю строку второй матрицы.

1.3 Параллельный алгоритм Винограда

Трудоёмкость алгоритма Винограда имеет сложность $O(NMK)$ для умножения матриц $N \times M$ на $M \times K$. Чтобы улучшить алгоритм, следует распараллелить ту часть алгоритма, которая содержит 3 вложенных цикла.

Вычисление результата для каждой строки не зависит от результата выполнения умножения для других строк. Поэтому можно распараллелить часть кода, где происходят эти действия. Каждый поток будет выполнять вычисления определенных строк результирующей матрицы.

1.4 Параллельное программирование

При использовании многопроцессорных вычислительных систем с общей памятью обычно предполагается, что имеющиеся в составе системы процессоры обладают равной производительностью, являются равноправными при доступе к общей памяти, и время доступа к памяти одинаково (при одновременном доступе нескольких процессоров к одному и тому же элементу памяти очередность и синхронизация доступа обеспечивается на аппаратном уровне). Многопроцессорные системы подобного типа обычно именуются симметричными мультипроцессорами.

Перечисленному выше набору предположений удовлетворяют также активно развиваемые в последнее время многоядерные процессоры, в которых каждое ядро представляет практически независимо функционирующее вычислительное устройство.

Обычный подход при организации вычислений для многопроцессорных вычислительных систем с общей памятью — создание новых параллельных методов на основе обычных последовательных программ, в которых или автоматически компилятором, или непосредственно программистом выделяются участки независимых друг от друга вычислений. Возможности автоматического анализа программ для порождения параллельных вычислений достаточно ограничены, и второй подход является преобладающим. При этом для разработки параллельных программ могут применяться как новые алгоритмические языки, ориентированные на параллельное программирование, так и уже имеющиеся

языки, расширенные некоторым набором операторов для параллельных вычислений.[2]

1.4 Организация взаимодействия параллельных потоков

Потоки исполняются в общем адресном пространстве параллельной программы. Как результат, взаимодействие параллельных потоков можно организовать через использование общих данных, являющимися доступными для всех потоков. Наиболее простая ситуация состоит в использовании общих данных только для чтения. В случае же, когда общие данные могут изменяться несколькими потоками, необходимы специальные действия для организации правильного взаимодействия.

Вывод

Был рассмотрен алгоритм Винограда и возможность его оптимизации с помощью распараллеливания потоков. Была рассмотрена технология параллельного программирования и организация взаимодействия параллельных потоков.

2. Конструкторская часть

В данном разделе будут рассмотрена схема улучшенного алгоритма Винограда и описание распараллеливания этого алгоритма.

2.1 Разработка алгоритмов

На рис. 1 приведена схема усовершенствованного алгоритма Винограда по умножению матриц.

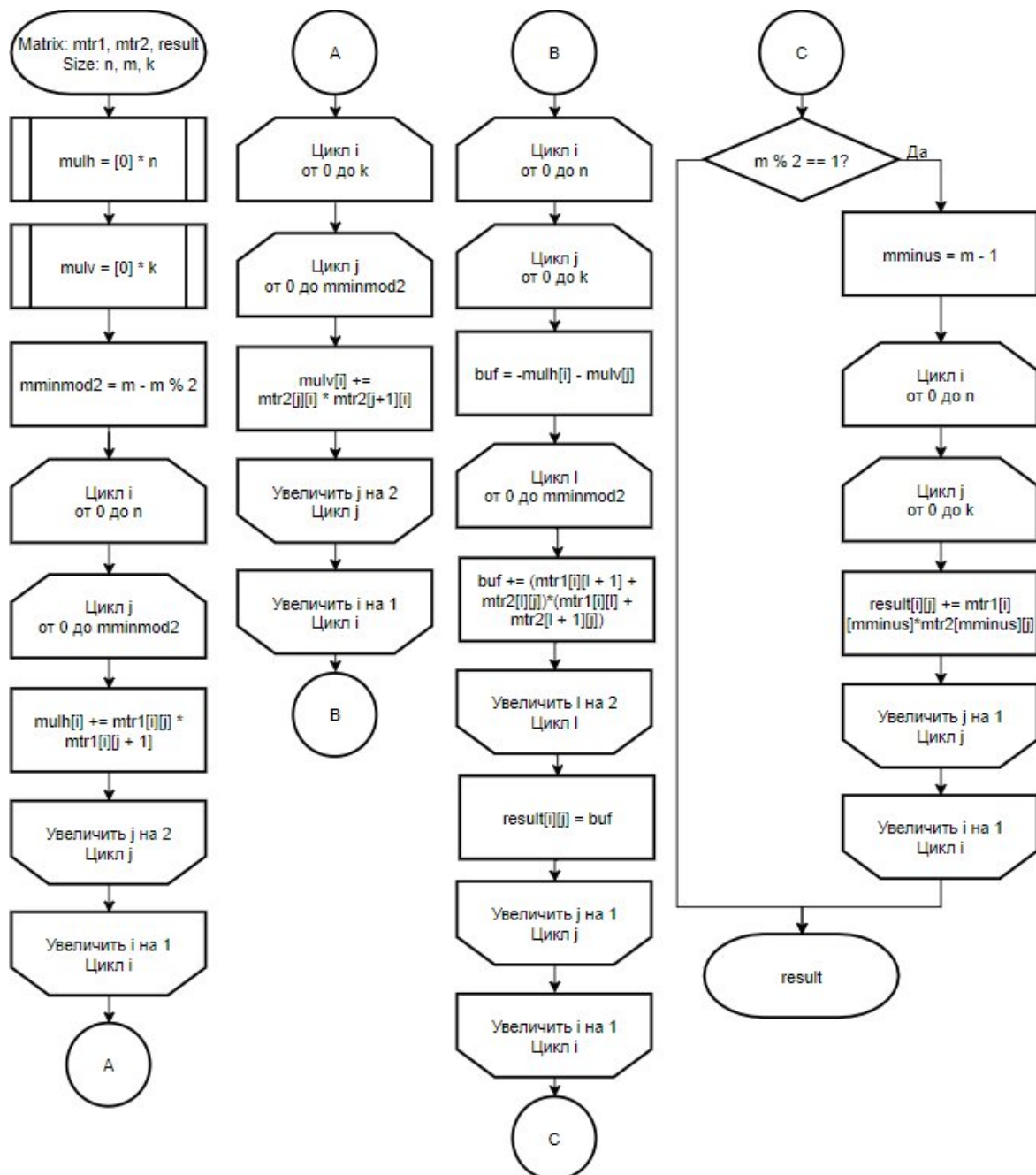


Рисунок 1. Схема усовершенствованного алгоритма Винограда по умножению матриц

2.2 Оптимизации алгоритма Винограда

Ниже представлены оптимизации для алгоритма Винограда, позволяющие уменьшить трудоемкость алгоритма.

- 1) Заменить конец цикла j с $m//2$, на переменную, инициализированную заранее $mminmod2 = m - m \% 2$ и $j += 1$ изменить на $j += 2$. Таким образом можно избавиться от постоянного деления в цикле.
- 2) Заменить все выражения типа $mulh[i] = mulh[i] + \dots$ на $mulh[i] +=$
- 3) Внутри тройного цикла накапливать результат в буфер, а вне цикла сбрасывать буфер в ячейку матрицы.
- 4) Заменить $m - 1$ в цикле, работающем при нечетном количестве столбцов первой матрицы, на переменную $mminus1 = m - 1$. Таким образом можно избавиться от постоянного вычитания в цикле.

2.3 Распараллеливание алгоритма Винограда

Распараллеливание программы должно ускорять время работы. Это достигается за счет реализации в узких участках (например, в циклах с большим количеством независимых вычислений).

В предложенном алгоритме данными участками являются тройной цикл поиска результата, цикл вычисления сумм перемноженных пар строк первой матрицы и сумм перемноженных пар столбцов второй матрицы.

Данные участки программы и будут распараллелены. Задача эффективного распараллеливания алгоритма может быть решена по-разному. Ниже представлены два варианта распараллеливания алгоритма умножения матриц по Винограду.

2.3 Распараллеливание по группам строк

Первая версия параллельного алгоритма заключается в том, чтобы разбить строки первой матрицы на блоки с одинаковым количеством строк в каждой, за исключением, когда количество строк не кратно количеству потоков, последнего. Таким образом мы получим схему, в которой каждый поток выполняет вычисления для каждой строки из своего блока, как это показано на рис. 2 (количество потоков равно 3). На нем видно, что матрица A разбита на блоки по две строки в каждом, кроме последнего, так как количество строк оказалось нечетным. Внутри каждого потока идет перемножение каждой строки, на каждый столбец матрицы B , после

выполнения работы каждым из потоков на выходе получается готовая матрица.

A

5	10	8	7
6	0	4	8
3	2	1	7
8	1	4	7
9	1	4	4

B

8	9	6
4	8	3
0	1	5
3	5	4

Рисунок 2. Версия параллельного алгоритма с разбивкой строк первой матрицы на блоки. В матрице A каждый цвет обозначает группу строк, принадлежащих отдельному потоку

При такой схеме нет проблем с обеспечением синхронизации, так как каждый блок независим от другого, и вычисления в одном никак не влияют на вычисления в другом.

2.4 Распараллеливание по строке

Данный подход заключается в том, чтобы каждый поток занимался обработкой лишь одной строки. В отличие от предыдущего варианта, он будет сложнее в реализации, в силу необходимости обеспечения синхронизации между потоками. На вход подается определенное количество потоков, которое необязательно будет равно количеству строк в таблице, следовательно возникает следующая проблема: как поток, после обработки строки, будет понимать какую следующую строчку ему обрабатывать? Для решения этого вопроса необходимо создать очередь, в которой будут лежать индексы тех строк, которые еще не были обработаны. Каждый поток будет обращаться к этой очереди и брать себе задачу, которую нужно обработать, после чего повторить эти действия. Разделение первой матрицы 2 потоками представлено на рис. 3. Несмотря на то, что на нем каждая строка закрашена одним из двух цветов (так как в данном примере только 2 потока), эти строки не обрабатываются сразу. Сначала 1 и 2 поток возьмут индексы 1 и 2 строки соответственно, произведут над ними вычисления, а затем перейдут к строке 3 и 4 и т.д.

А

5	10	8	7
6	0	4	8
3	2	1	7
8	1	4	7
9	1	4	4

В

8	9	6
4	8	3
0	1	5
3	5	4

Рисунок 3. Версия параллельного алгоритма с построчной обработкой матрицы каждым потоком. В матрице А каждый цвет обозначает строку, которая будет обработана конкретным потоком.

Проблема заключается в захвате индекса следующей строки из очереди. Потоки работают внутри одного процесса, а следовательно разделяют его общую память. Если один поток получит значение из очереди, то другой может просто “не увидеть” этого и обработать эту же строку, что приведет к неверному ответу. Для того чтобы синхронизировать потоки, необходимо воспользоваться мьютексом. Мьютекс — примитив синхронизации, который позволяет захватить владение каким-либо образом повлиять на данный объект, пока данный поток не вернет его в общее пользование. [3]

Вывод

В данном разделе была построена схема алгоритма усовершенствованного умножения матриц Винограда. Также были описано распараллеливание данного алгоритма.

3. Технологическая часть

В данном разделе будут рассмотрены требования к программному обеспечению, средства реализации и представлен листинг кода.

3.1 Требования к программному обеспечению

Входные данные: mtr1 – первая матрица, mtr2 – вторая матрица.

Выходные данные: произведение двух матриц.

Функциональная схема процесса перемножения двух матриц в нотации IDEF0 представлена на рис. 2.

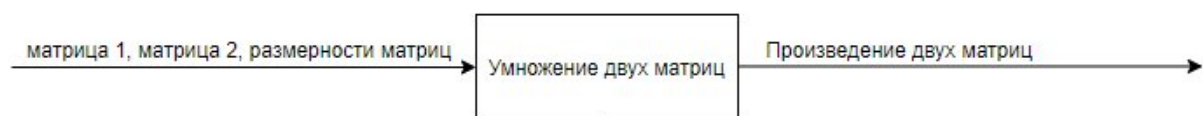


Рисунок 4. Функциональная схема процесса перемножения двух матриц

3.2 Средства реализации

В качестве языка программирования был выбран C++ т.к. данный язык часто использовался мною. Данный язык позволяет писать высокоуровневый абстрактный код, который при этом работает со скоростью близкой к машинному коду.

Среда разработки – Visual Studio, которая предоставляет умную проверку кода, быстрое выявление ошибок и оперативное исправление, вкупе с автоматическим рефакторингом кода и богатыми возможностями в навигации.

Время работы алгоритмов было замерено с помощью функции `steady_clock()` из библиотеки `chrono`. [4]

Для тестирования использовался компьютер на базе процессора AMD A8-7410 APU with AMD Radeon R5 Graphics, 2200 МГц, ядер: 4, логических процессоров: 4.

Многопоточное программирование было реализовано с помощью библиотеки `thread`. [5]

3.3 Листинг кода

В данном пункте представлен листинг кода функций. В листинге 1 представлена реализация улучшенного алгоритма умножения матриц по Винограду. В листингах 2 - 5 представлена реализация распараллеленного по

группам строк улучшенного алгоритма Винограда. В листингах 6 - 7 представлена реализация распараллеленного по строке улучшенного алгоритма Винограда.

Листинг 1. Функция улучшенного алгоритма Винограда

```
int vinogradOPT(const vector<vector<int>>& A, const vector<vector<int>>& B,
    const int m, const int n, const int q, vector<vector<int>>& C){
    vector<int> mulH(m, 0);
    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < n - 1; j += 2)
        {
            mulH[i] -= A[i][j] * A[i][j + 1];
        }
    }

    vector<int> mulV(q, 0);
    for (int i = 0; i < q; i++)
    {
        for (int j = 0; j < n - 1; j += 2)
        {
            mulV[i] -= B[j][i] * B[j + 1][i];
        }
    }

    int last = n - 1;
    bool flag = n % 2 == 1;
    C = vector<vector<int>>(m, vector<int>(q, 0));
    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < q; j++)
        {
            int tmp = mulH[i] + mulV[j];
            for (int k = 0; k < n - 1; k += 2)
            {
                tmp += (A[i][k] + B[k + 1][j]) * (A[i][k + 1] + B[k][j]);
            }
        }
    }
}
```

```

        if (flag)
        {
            tmp += A[i][last] * B[last][j];
        }
        C[i][j] = tmp;
    }
}
return OK;}

```

Листинг 2. Функция многопоточного улучшенного алгоритма Винограда (по группам строк)

```

int threadedVinogradOPT(const vector<vector<int>>& A, const
vector<vector<int>>& B,
    const int m, const int n, const int q, vector<vector<int>>& C,
    const int& nThreads){
    vector<thread> threads;
    vector<int> mulH(m, 0);
    double start = 0;
    double del = m / static_cast<double>(nThreads);
    for (int i = 0; i < nThreads; i++)
    {
        threads.push_back(thread(computeMulH, ref(mulH), A, round(start),
            round(start + del)));
        start += del;
    }
    for (auto& thread : threads)
    {
        thread.join();
    }
    start = 0;
    del = q / static_cast<double>(nThreads);
    vector<int> mulV(q, 0);
    for (int i = 0; i < nThreads; i++)
    {
        threads[i] = thread(computeMulV, ref(mulV), B, round(start),
            round(start + del));
        start += del;
    }
    for (auto& thread : threads)

```

```

{
    thread.join();
}

C = vector<vector<int>>(m, vector<int>(q, 0));
start = 0;
del = m / static_cast<double>(nThreads);
for (int i = 0; i < nThreads; i++)
{
    threads[i] = thread(computeResult, ref(C), A, B,
        mulH, mulV, round(start), round(start + del));
    start += del;
}
for (auto& thread : threads)
{
    thread.join();
}
return OK;}

```

Листинг 3. Функция вычисления сумм строк первой матрицы

```

void computeMulH(vector<int>& mulH, vector<vector<int>> A, int startRow, int
endRow){
    int n = A[0].size();
    for (int i = startRow; i < endRow; i++)
    {
        for (int j = 0; j < n - 1; j += 2)
        {
            mulH[i] -= A[i][j] * A[i][j + 1];
        }
    }
}

```

Листинг 4. Функция вычисления сумм столбцов второй матрицы

```

void computeMulV(vector<int>& mulV, vector<vector<int>> B, int startCol, int
endCol){
    int n = B.size();
    for (int i = startCol; i < endCol; i++)
    {
        for (int j = 0; j < n - 1; j += 2)
        {
            mulV[i] -= B[j][i] * B[j + 1][i];
        }
    }
}

```

```

    }
}

```

Листинг 5. Функция вычисления результирующей матрицы

```

void computeResult(vector<vector<int>>& C, vector<vector<int>> A,
    vector<vector<int>> B, vector<int> mulH,
    vector<int> mulV, int startRow, int endRow){
    int n = B.size();
    int q = B[0].size();
    int last = n - 1;
    bool flag = n % 2 == 1;
    for (int i = startRow; i < endRow; i++)
    {
        for (int j = 0; j < q; j++)
        {
            int tmp = mulH[i] + mulV[j];
            for (int k = 0; k < n - 1; k += 2)
            {
                tmp += (A[i][k] + B[k + 1][j]) * (A[i][k + 1] + B[k][j]);
            }
            if (flag)
            {
                tmp += A[i][last] * B[last][j];
            }
            C[i][j] = tmp;
        }
    }
}

```

Листинг 6. Функция многопоточного улучшенного алгоритма Винограда (по строке)

```

int threadedVinogradOPT2(const vector<vector<int>>& A, const
vector<vector<int>>& B,
    const int m, const int n, const int q, vector<vector<int>>& C,
    const int& nThreads){
    vector<thread> threads;
    vector<int> mulH(m, 0);
    double start = 0;
    double del = m / static_cast<double>(nThreads);
    for (int i = 0; i < nThreads; i++)
    {

```

```

        threads.push_back(thread(computeMulH, ref(mulH), A, round(start),
                                round(start + del)));
        start += del;
    }
    for (auto& thread : threads)
    {
        thread.join();
    }
    start = 0;
    del = q / static_cast<double>(nThreads);
    vector<int> mulV(q, 0);
    for (int i = 0; i < nThreads; i++)
    {
        threads[i] = thread(computeMulV, ref(mulV), B, round(start),
                            round(start + del));
        start += del;
    }
    for (auto& thread : threads)
    {
        thread.join();
    }

    C = vector<vector<int>>(m, vector<int>(q, 0));
    queue<int> que;

    for (int i = 0; i < m; i++)
        que.push(i);
    for (int i = 0; i < nThreads; i++)
    {
        threads[i] = thread(computeResult1, ref(C), A, B,
                            mulH, mulV, ref(que));
    }
    for (auto& thread : threads)
    {
        thread.join();
    }
    return OK;}

```


Листинг 7. Функция вычисления результирующей матрицы

```
void computeResult1(vector<vector<int>>& C, vector<vector<int>> A,
    vector<vector<int>> B, vector<int> mulH,
    vector<int> mulV, queue<int> &que){
    int n = B.size();
    int q = B[0].size();
    int last = n - 1;
    bool flag = n % 2 == 1;
    mutex m;
    while (true)
    {
        m.lock();
        if (que.empty())
        {
            m.unlock();
            break;
        }
        int i = que.front();
        que.pop();
        m.unlock();
        for (int j = 0; j < q; j++)
        {
            int tmp = mulH[i] + mulV[j];
            for (int k = 0; k < n - 1; k += 2)
            {
                tmp += (A[i][k] + B[k + 1][j]) * (A[i][k + 1] + B[k][j]);
            }
            if (flag)
            {
                tmp += A[i][last] * B[last][j];
            }
            C[i][j] = tmp;
        }
    }
}
```

3.4 Тестирование

В тестах будет рассмотрена проверка работы алгоритмов при:

- нулевой матрице;
- единичной матрице;
- матриц, содержащих отрицательные числа;

Тестирование производится методом черного ящика.

На рисунках ниже будут приведены тесты с целью демонстрации корректности работы программы.

```
Matrix A
1 3 6
2 8 2
7 7 7
Matrix B
0 0 0
0 0 0
0 0 0
AXB result (Vinograd Opt)
0 0 0
0 0 0
0 0 0
AXB result (Vinograd Opt threaded 1)
0 0 0
0 0 0
0 0 0
AXB result (Vinograd Opt threaded 2)
0 0 0
0 0 0
0 0 0
```

Рисунок 5. Проверка работы алгоритмов при нулевой матрице

```
Matrix A
1 3 6
2 8 2
7 7 7
Matrix B
1 0 0
0 1 0
0 0 1
AXB result (Vinograd Opt)
1 3 6
2 8 2
7 7 7
AXB result (Vinograd Opt threaded 1)
1 3 6
2 8 2
7 7 7
AXB result (Vinograd Opt threaded 2)
1 3 6
2 8 2
7 7 7
```

Рисунок 6. Проверка работы алгоритмов при единичной матрице

```
Matrix A
1 -3 6
-2 8 -2
7 -7 7
Matrix B
4 3 5
2 1 -3
1 -1 -1
AXB result (Vinograd Opt)
4 -6 8
6 4 -32
21 7 49
AXB result (Vinograd Opt threaded 1)
4 -6 8
6 4 -32
21 7 49
AXB result (Vinograd Opt threaded 2)
4 -6 8
6 4 -32
21 7 49
```

Рисунок 7. Проверка работы алгоритмов при матрицах, содержащих отрицательные числа

Все тесты пройдены успешно.

Вывод

В данном разделе была представлена реализация алгоритмов улучшенного Винограда, а также версия с параллельными вычислениями улучшенного Винограда. Также проведено тестирование разработанных методов по методу чёрного ящика.

4. Экспериментальная часть

В данном разделе будет проведен эксперимент и сравнительный анализ полученных данных.

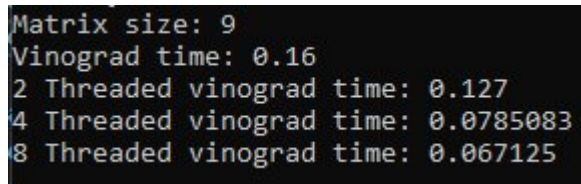
4.1 Постановка эксперимента

В рамках данной части были проведены эксперименты, описанные ниже.

- 1) Сравнение времени работы двух алгоритмов, где алгоритмы, использующие распараллеливание, тестировались при 2, 4 и 8 потоках. Создаются квадратные матрицы размерностью 9 (нечетный размер), заполненные случайными числами. Эксперимент ставился 1000 раз;
- 2) Сравнение времени работы двух алгоритмов, где алгоритмы, использующие распараллеливание, тестировались при 2, 4 и 8 потоках. Создаются квадратные матрицы размерностью 10 (четный размер), заполненные случайными числами. Эксперимент ставился 1000 раз.

4.2 Сравнительный анализ на материале экспериментальных данных

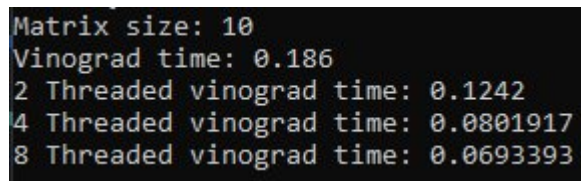
Сравнение времени работы алгоритмов при квадратных матрицах размерностью 9 и количестве итераций 1000 изображены на рис. 9.



```
Matrix size: 9
Vinograd time: 0.16
2 Threaded vinograd time: 0.127
4 Threaded vinograd time: 0.0785083
8 Threaded vinograd time: 0.067125
```

Рисунок 9. Время работы алгоритмов при квадратных матрицах размерностью 9

Сравнение времени работы алгоритмов при квадратных матрицах размерностью 10 и количестве итераций 1000 изображены на рис. 11



```
Matrix size: 10
Vinograd time: 0.186
2 Threaded vinograd time: 0.1242
4 Threaded vinograd time: 0.0801917
8 Threaded vinograd time: 0.0693393
```

Рисунок 11. Время работы алгоритмов при квадратных матрицах размерностью 10

Вывод

В данном разделе был поставлен эксперимент по замеру времени выполнения улучшенного алгоритма Винограда с применением распараллеливания (двумя вариантами) и без. По итогам замеров можно сделать вывод о том, что оба варианта улучшенного алгоритма Винограда с применением распараллеливания работают быстрее при увеличении числа потоков, а также работает быстрее чем улучшенный алгоритм Винограда. Однако установлено, что увеличение количества потоков имеет смысл, пока не будет достигнуто число, равное количеству логических процессоров в системе, причем самой быстрой версией параллельного алгоритма Винограда является та, где число потоков равно числу логических процессоров.

Заключение

В ходе работы были изучены возможности параллельных вычислений, а также были использованы на практике. Был реализован улучшенный алгоритм Винограда с использованием параллельных вычислений и без. Также были произведены замеры времени выполнения алгоритмов, в котором улучшенный алгоритм Винограда с использованием распараллеливания оказался быстрее реализации без распараллеливания. Также было выявлено, что увеличение количества потоков имеет смысл, пока не будет достигнуто число, равное количеству логических процессоров в системе, причем максимальная скорость работы достигается при количестве потоков равному количеству логических процессоров. Все поставленные задачи были выполнены. Целью лабораторной работы являлось изучение параллельных вычислений, что также было достигнуто.

Список использованных источников

1. Курош А. Г. Курс высшей алгебры — 9-е изд. — М.: Наука, 1968. — 432 с.
2. Константин Баркалов, Владимир Воеводин, Виктор Гергель. Intel Parallel Programming [электронный ресурс]. Режим доступа: <https://www.intuit.ru/studies/courses/4447/983/lecture/14925>, свободный (Дата обращения: 10.11.20)
3. Воеводин В. В., Воеводин Вл. В. Параллельные вычисления. — СПб: БХВ-Петербург, 2002. — 608 с.
4. Официальный сайт Microsoft, документация [электронный ресурс]. Режим доступа: <https://docs.microsoft.com/ru-ru/cpp/standard-library/chrono?view=vs-2017>, свободный (Дата обращения: 10.11.20)
5. Официальный сайт Microsoft, документация [электронный ресурс]. Режим доступа: <https://docs.microsoft.com/ru-ru/cpp/standard-library/thread-class?view=vs-2019>, свободный (Дата обращения: 10.11.20)