Государственное образовательное учреждение высшего
профессионального образования
“Московский государственный технический университет имени
Н.Э.Баумана”

Дисциплина: Анализ алгоритмов

Лабораторная работа № 2

Многопоточное программирование

Гибадулин О.Н.
Студент группы ИУ7-52

2019 г.

Содержание

Введение

В настоящие время практически все устройства – от карманных гаджетов и до самых мощных суперкомпьютеров – оснащены многоядерными процессорами. С переходом компьютерной индустрии на массовый выпуск многоядерных процессоров широкое распространение приобрел принцип параллельных вычислений.

В данной работе требуется рассмотреть алгоритм Винограда для умножения матриц в однопоточной и многопоточной реализациях, а также провести их сравнительный анализ.

Цель работы: изучение многопоточности и её использование на примере алгоритма Винограда.

Задачи работы:

1. разработка и реализация алгоритмов;
2. исследование временных затрат алгоритмов;
3. описание и обоснование полученных результатов.

1 Аналитический раздел

В данном разделе будет описан алгоритм Винограда перемножения матриц.

1.1 Алгоритм Винограда

Алгоритм Винограда это улучшенная версия стандартного алгоритма, где часть вычислений производится заранее. Рассмотрим стандартный алгоритм перемножения двух матриц. Пусть есть две матрицы A и B размера $a \cdot b$ и $c \cdot d$ соответственно. Тогда, результатом из умножения будет матрица C размером $a \cdot d$, имеющая вид(1):

$$\begin{bmatrix} c_{11} & c_{12} & \dots & c_{1d} \\ c_{21} & c_{22} & \dots & c_{2d} \\ \dots & \dots & \dots & \dots \\ c_{a1} & c_{a2} & \dots & c_{ad} \end{bmatrix} \tag{1}$$

Каждый элемент матрицы (1) представляет собой скалярное произведение соответствующих строки и столбца исходных матриц.

Рассмотрим два вектора:

$$V = (v_1, v_2, v_3, v_4) \tag{2}$$

и

$$W = (w_1, w_2, w_3, w_4) \tag{3}$$

Их скалярное произведение:

$$V * W = v_1 \cdot w_1 + v_2 \cdot w_2 + v_3 \cdot w_3 + v_4 \cdot w_4. \tag{4}$$

Это равенство можно переписать в виде:

$$V*W = (v_1+w_2)\cdot(v_2+w_1)+(v_3+w_4)\cdot(v_4+w_3)-v_1\cdot v_2-v_3\cdot v_4-w_1\cdot w_2-w_3\cdot w_4 \tag{5}$$

Несмотря на то, что выражение (5) требует больше вычисления, чем (4), выражение в правой части последнего равенства (5) допускает предварительную обработку. Части этого выражения 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. В этом и заключается алгоритм Винограда.[1]

1.2 Многопоточность

Существуют зеленые и нативные потоки. Зелёные потоки — это потоки выполнения, управление которыми вместо операционной системы выполняет виртуальная машина. Зелёные потоки эмулируют многопоточную среду, не полагаясь на возможности ОС по реализации легковесных потоков. Управление ими происходит в пользовательском пространстве, а не пространстве ядра, что позволяет им работать в условиях отсутствия поддержки встроенных потоков. На многоядерных процессорах родная реализация нативных потоков может автоматически назначать работу нескольким процессорам, в то время как реализация зелёных потоков обычно не может. Зелёные потоки могут быть запущены гораздо быстрее на некоторых виртуальных машинах.

Поток выполнения — это 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[2].

1.3 Вывод

В данном разделе был описан алгоритм Винограда перемножения матриц.

2 Конструкторский раздел

В данном разделе будет формализован и описан процесс вычисления произведения двух матриц с помощью диаграммы idef0, а также в соответствии с описанием алгоритмов, приведенными в аналитической части работы, будут рассмотрены схемы алгоритма Винограда перемножения матриц.

2.1 Формализация процесса

В данном пункте представлена idef0-диаграмма для описания функциональной модели процесса вычисления произведения двух матриц (рис. 2.1.1, рис. 2.1.2).

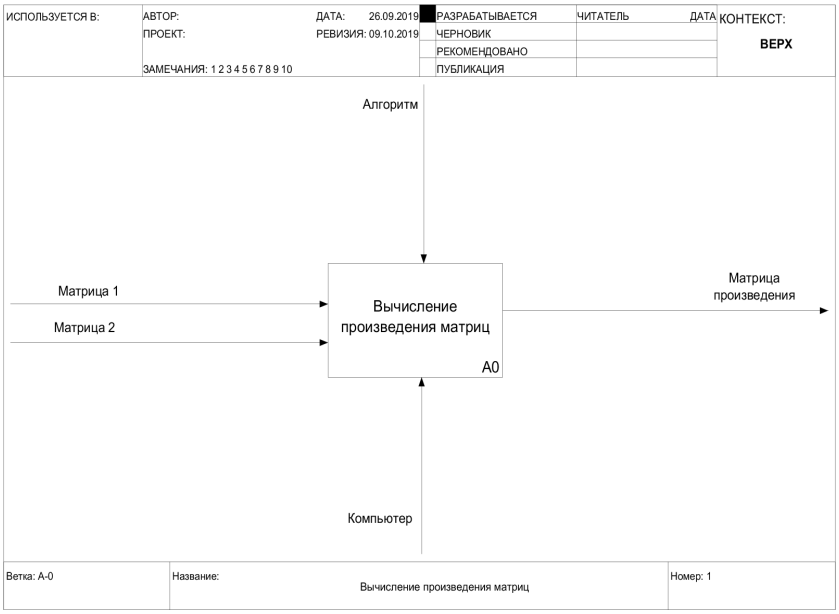

Рисунок 2.1.1. – функциональная схема верхнего уровня процесса умножения матриц

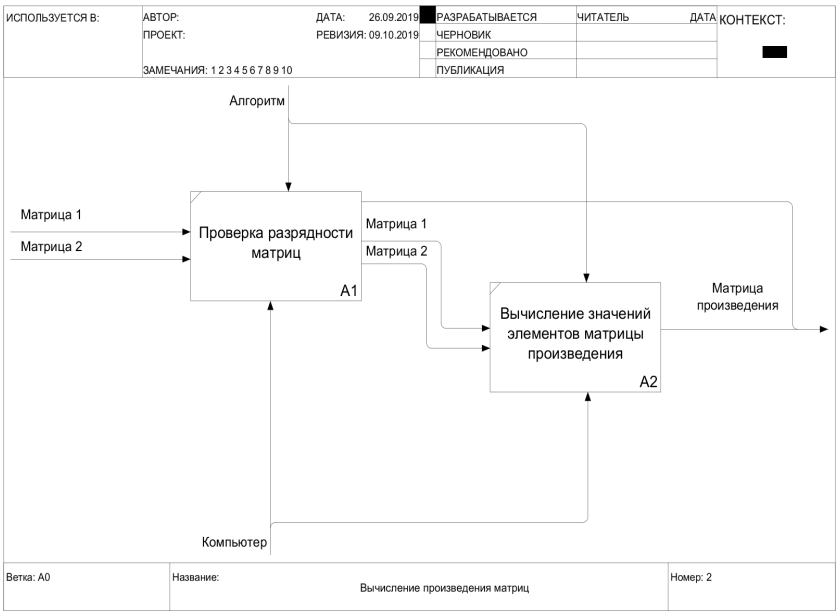

Рисунок 2.1.2. – функциональная схема второго уровня процесса умножения матриц

2.2 Разработка Алгоритма

В данном пункте представлена схема алгоритма Винограда. Алгоритм начинается с проверки разрядности перемножаемых матриц. Если количество строк или столбцов хотя бы одной из матриц равно 0 или количество столбцов первой не равно количеству строк во второй, то выходим из алгоритма, возвращая нулевую матрицу как матрицу произведения.

В алгоритме Винограда (рис. 2.2.1, рис. 2.2.2) для каждой строки первой матрицы (A) вычисляются и заносятся в массив RF произведения элементов, стоящих на чётных и нечётных позициях. Аналогично вычисляются и заносятся в массив CF произведения значений, находящихся в каждой колонке второй матрицы (B).

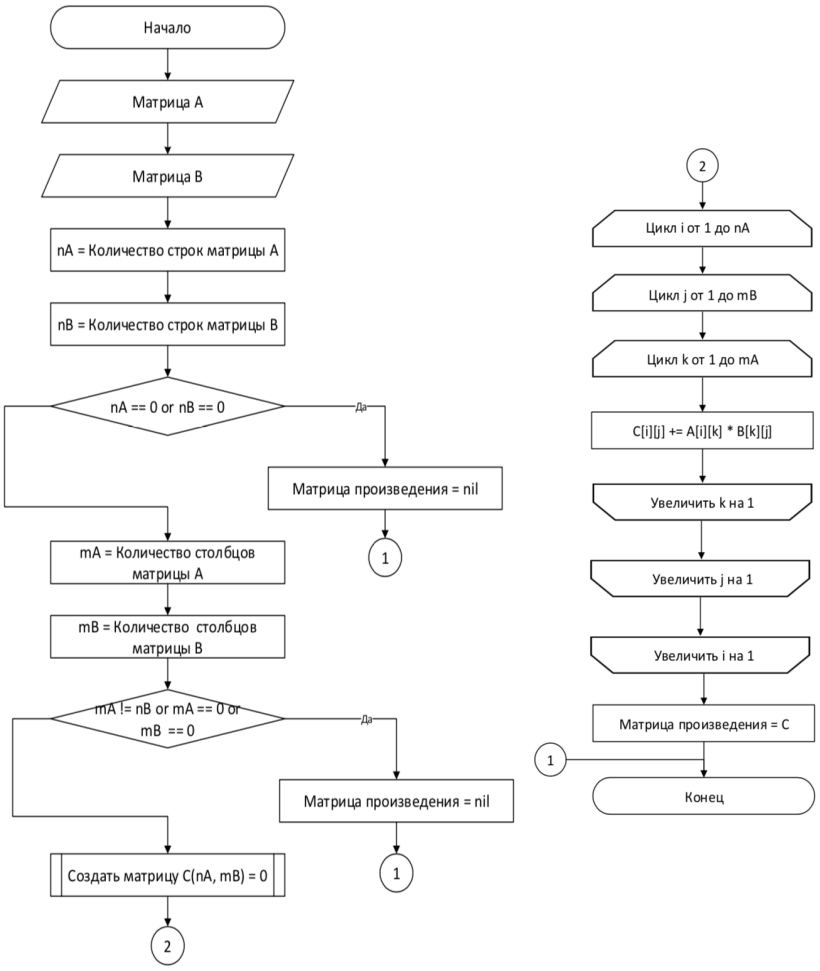

Рисунок 2.2.1. – схема алгоритма Винограда

Далее в алгоритме Винограда для каждой ячейки конечной матрицы (C) вычисляется сумма произведений сумм чётных элементов с нечётными из столбцов первой матрицы (A) и строк второй матрицы (B). Также для матриц с нечётным значением размерности ко всем элементам конечной матрицы прибавляется произведение значений элементов из последнего столбца первой матрицы (A) и значений элементов последней строки второй матрицы (B).

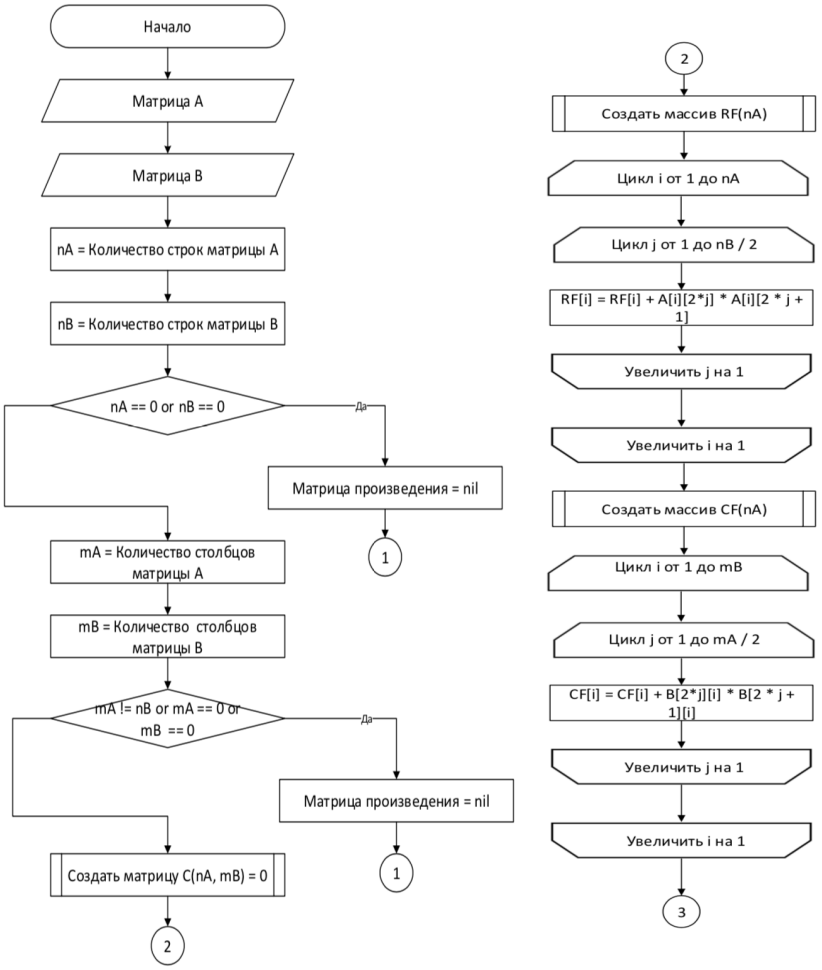

Рисунок 2.2.2. – схема алгоритма Винограда

Для реализации многопоточной версии алгоритма Винограда выделим 5 основных частей:

1. создание и инициализация результирующей матрицы произведения;
2. создание и инициализация rowFactor;
3. создание и инициализация columnFactor;
4. основные вычисление матрицы произведения;
5. дополнительные вычисления при нечетной размерности входных матриц.

Вычисление матрицы прозведения возможно после выполнения 1, 2 и 3 пунктов, поэтому для каждого из них выделяется поток, который должен завершится до выполнения 4 и 5 пунктов. Для основных вычислений матрицы произведения выделяется несколько потоков. Если размерность входных матриц является нечетной, то также выделится поток для дополнительных вычислений, которые будут производится параллельно с вычислением основной части.

2.3 Вывод

В данном разделе был формализован и описан процесс вычисления произведения двух матриц, а также рассмотрены схемы алгоритма Винограда и стандартного алгоритма перемножения матриц.

3 Технологический раздел

В данном разделе будут рассмотрены требования к разрабатываемому программному обеспечению, средства, использованные в процессе разработки для реализации поставленных задач, а также представлен листинг кода программы.

3.1 Требования к программному обеспечению

Программное обеспечение должно реализовывать алгоритм Винограда перемножения двух матриц в однопоточной и многопоточной реализациях. Пользователь должен иметь возможность произвести вычисления для матриц, размер которых он вводит, а также иметь возможность сравнить время работы этих алгоритмов.

3.2 Средства реализации

Для реализации поставленной задачи был использован язык программирования C++[3], поддерживающий нативные потоки. Для реализации многопоточности была использована стандратная библиотека thred[3]. Проет был выполнен в среде XCode[4]. Для измерения времени в милисекундах была использована библиотека chrono[3].

3.3 Листинг кода

На основе схем алгоритмов, представленных в конструкторском разделе, в соответствии с указанными требованиями к реализации с использованием средств языка C++ было разработано программное обеспечение, содержащее реализации выбранных алгоритмов. В данном пункте приведён листиннг этих реализаций.

Листинг 1. Код однопоточной реализации алгоритма Винограда

```
        const size_t nA = matrixA.size();
        const size_t nB = matrixB.size();
        if (nA == 0 || nB == 0)
        {
        cout << "Wrong dimension of the matrix" << endl;
        return vector<vector<int>>();
        }

        const size_t mA = matrixA[0].size();
        const size_t mB = matrixB[0].size();
        if (mA != nB || mA == 0 || mB == 0)
        {
        cout << "Wrong dimension of the matrix" << endl;
        return vector<vector<int>>();
        }
```

```
vector<vector<int>> matrixRes(nA);
for (int i = 0; i < nA; ++i)
{
matrixRes[i].assign(mB, 0);
}

vector<int> rowFactor(nA);
for (int i = 0; i < nA; ++i)
{
for (int j = 0; j < nB / 2; ++j)
{
rowFactor[i] = rowFactor[i] + (matrixA[i][2 * j]
    * matrixA[i][2 * j + 1]);
}
}

vector<int> columnFactor(mB);
for (int i = 0; i < mB; ++i)
{
for (int j = 0; j < mA / 2; ++j)
{
columnFactor[i] = columnFactor[i] + (matrixB[2 *
    j][i] * matrixB[2 * j + 1][i]);
}
}

for (int i = 0; i < nA; ++i)
{
for (int j = 0; j < mB; ++j)
{
matrixRes[i][j] = -rowFactor[i] - columnFactor[j
    ];
for (int k = 0; k < mA / 2; ++k)
{
matrixRes[i][j] = matrixRes[i][j] + ((matrixA[i
    ][2 * k] + matrixB[2 * k + 1][j]) * (matrixA[i
    ][2 * k + 1] + matrixB[2 * k][j]));
}
}
}

if (mA % 2 != 0)
{
for (int i = 0; i < nA; ++i)
{
for (int j = 0; j < mB; ++j)
{
matrixRes[i][j] = matrixRes[i][j] + (matrixA[i][
    mA - 1] * matrixB[nB - 1][j]);
}
```

```
        }
        }

        return matrixRes;
```

Листинг 2. Код многопоточной реализации алгоритма Винограда

```
        void getMatrixRes(vector<vector<int>>& matrixRes,
             const size_t nA, const size_t mB) {
        vector<vector<int>> tmp(nA);
        for (size_t i = 0; i < nA; ++i) {
        tmp[i].assign(mB, 0);
        }
        matrixRes = tmp;
        }

        void calculateRowFactor(vector<int>& rowFactor,
            const vector<vector<int>>& matrixA, const
            vector<vector<int>>& matrixB, const size_t nA,
             const size_t d) {
        vector<int> tmp(nA);
        for (size_t i = 0; i < nA; ++i) {
        for (size_t j = 0; j < d; ++j) {
        tmp[i] += (matrixA[i][(j * 2)] * matrixA[i][(j *
            2) + 1]);
        }
        }

        rowFactor = tmp;
        }

        void calculateColumnFactor(vector<int>&
            columnFactor, const vector<vector<int>>&
            matrixA, const vector<vector<int>>& matrixB,
            const size_t mB, const size_t d) {
        vector<int> tmp(mB);
        for (size_t i = 0; i < mB; ++i) {
        for (size_t j = 0; j < d; ++j) {
        tmp[i] += (matrixB[(j * 2)][i] * matrixB[(j * 2)
            + 1][i]);
        }
        }
        columnFactor = tmp;
        }

        void calculateGeneralProductionRows(vector<vector
            <int>>& matrixRes, const vector<vector<int>>&
            matrixA, const vector<vector<int>>& matrixB,
            const vector<int>& rowFactor, const vector<int
            >& columnFactor, const size_t lowRow, const
```

```
    size_t highRow, const size_t mB, const size_t
    d) {
for (size_t i = lowRow; i < highRow; ++i) {
for (size_t j = 0; j < mB; ++j) {
matrixRes[i][j] = -rowFactor[i] - columnFactor[j
    ];
for (size_t k = 0; k < d; ++k) {
matrixRes[i][j] = matrixRes[i][j] + ((matrixA[i
    ][2 * k] + matrixB[2 * k + 1][j]) * (matrixA[i
    ][2 * k + 1] + matrixB[2 * k][j]));
}
}
}
}

void calculateAdditionalProduction(vector<vector<
    int>>& matrixRes, const vector<vector<int>>&
    matrixA, const vector<vector<int>>& matrixB,
    const size_t nA, const size_t mB, const size_t
     nB, const size_t mA) {
for (size_t i = 0; i < nA; ++i) {
for (size_t j = 0; j < mB; ++j) {
matrixRes[i][j] = matrixRes[i][j] + (matrixA[i][
   mA - 1] * matrixB[nB - 1][j]);
}
}
}

vector<vector<int>> parallelMultiplyWinograd(
    const vector<vector<int>>& matrixA, const
    vector<vector<int>>& matrixB, const size_t
    threadsCount = 4)
{
const size_t nA = matrixA.size();
const size_t nB = matrixB.size();
if (nA == 0 || nB == 0) {
cout << "Wrong dimension of the matrix" << endl;
return vector<vector<int>>();
}

const size_t mA = matrixA[0].size();
const size_t mB = matrixB[0].size();
if (mA != nB || mA == 0 || mB == 0) {
cout << "Wrong dimension of the matrix" << endl;
return vector<vector<int>>();

}

vector<vector<int>> matrixRes;
vector<int> rowFactor;
```

```
vector<int> columnFactor;
const size_t d = nB / 2;

thread matrixResThread(getMatrixRes, ref(
    matrixRes), nA, mB);
thread rowFactorThread(calculateRowFactor, ref(
    rowFactor), ref(matrixA), ref(matrixB), nA, d)
    ;
thread columnFactorThread(calculateColumnFactor,
    ref(columnFactor), ref(matrixA), ref(matrixB),
     mB, d);

if (matrixResThread.joinable() && rowFactorThread
    .joinable() && columnFactorThread.joinable())
    {
matrixResThread.join();
rowFactorThread.join();
columnFactorThread.join();
}

double threadRowCount = (double) nA /
    threadsCount;
if (threadRowCount < 1) {
threadRowCount = 1;
}
vector<thread> threads;

for (size_t i = 0, lowRow = 0, highRow = 0 &&
    highRow < nA; i < threadsCount; ++i) {
highRow += threadRowCount;
if (i == threadsCount) {
highRow = nA;
}
threads.push_back(thread(
    calculateGeneralProductionRows, ref(matrixRes)
    , ref(matrixA), ref(matrixB), ref(rowFactor),
    ref(columnFactor), lowRow, highRow, mB, d));
lowRow = highRow;
}

if (mA % 2 != 0) {
thread additionalProductionTread(
    calculateAdditionalProduction, ref(matrixRes),
     ref(matrixA), ref(matrixB), nA, mB, nB, mA);
if (additionalProductionTread.joinable()) {
additionalProductionTread.join();
}
}

for (int i = 0; i < threads.size(); ++i) {
```

```
        if (threads[i].joinable()) {
        threads[i].join();
        }
        }
        threads.clear();

        return matrixRes;
        }
```

3.4 Вывод

В данном разделе были рассмотрены требования к разрабатываемому программному обеспечению, средства, использованные в процессе разработки, а также был представлен листинг реализаций выбранных алгоритмов.

4 Экспериментальный раздел

В данном разделе будет проведено исследование временных затрат разработанного программного обеспечения, вместе с подробным сравнительным анализом реализованных алгоритмов на основе экспериментальных данных.

4.1 Сравнительное исследование

Для сравнения однопоточной и многопоточной версий алгоритмов замеры времени выполнялись на квадратных матрицах размера от 100x100 до 1000x1000 с шагом 100 для четной совпадающей размерности матриц (табл. 4.1.1, рис. 4.1.1), и от 101x101 до 1001x1001 для нечетной (табл. 4.1.2, рис. 4.1.2).

Для сравнения многопоточной версии алгоритма при различном количестве выделенных потоков замеры времени выполнялись на квадратных матрицах размера от 100x100 до 1000x1000 с шагом 100 при количестве потоков от 1 до 7 (табл. 4.1.3, рис. 4.1.3).

Числа в матрицах генерировались случайным образом. Все замеры были произведены на 2-х ядерном процессоре 2,7 GHz Intel Core i5 с паматью 8 ГБ 1867 MHz DDR3.

Таблица 4.1.1. Сравнение времени работы однопоточной и многопоточной версий алгоритмов в милисекундах для чётной размерности матриц

Размерность матриц	Однопоточный алг. Винограда	Многопоточный алг. Винограда
100	17	7
200	155	73
300	527	244
400	1519	603
500	3274	1376
600	5506	2126
700	9108	3350
800	14415	5404
900	19674	7608
1000	31607	11821

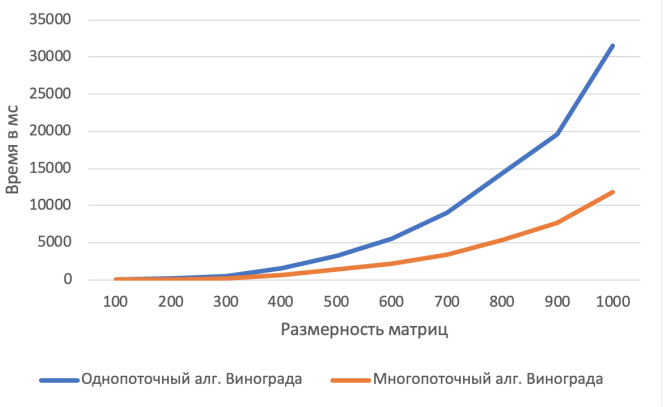

Рисунок 4.1.1. – график зависимости времени работы однопоточной и многопоточной версий алгоритмов для чётной размерности матриц

Таблица 4.1.2. Сравнение времени работы однопоточной и многопоточной версий алгоритмов в милисекундах для нечётной размерности матриц

Размерность матриц	Однопоточный алг. Винограда	Многопоточный алг. Винограда
101	18	8
201	148	71
301	586	256
401	1585	576
501	3158	1351
601	5481	2134
701	8553	3865
801	14433	5174
901	21187	7844
1001	40555	11392

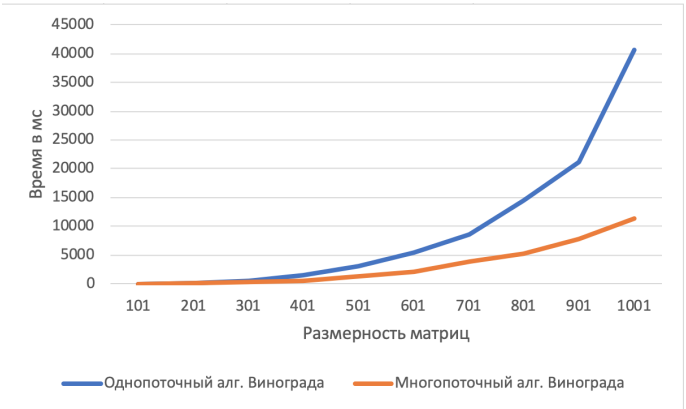

Рисунок 4.1.2. – график зависимости времени работы однопоточной и многопоточной версий алгоритмов для нечётной размерности матриц

Таблица 4.1.3. Сравнение времени работы многопоточной версии алгоритма в милисекундах при различном количестве выделенных потоков

Размерность матриц	1	2	3	4	5	6	7
100	15	7	8	8	9	8	9
200	129	65	62	61	62	61	64
300	427	222	221	212	213	213	235
400	1107	593	548	525	525	518	524
500	2235	1146	1100	1049	1036	1030	1077
600	3851	1990	1858	1809	1779	1781	1786
700	6170	3169	3007	2871	2861	2854	2888
800	9733	4906	4633	4386	4381	4382	4482
900	14947	8224	7753	7260	7254	7255	7369
1000	28486	15138	13125	12476	12441	12435	12696

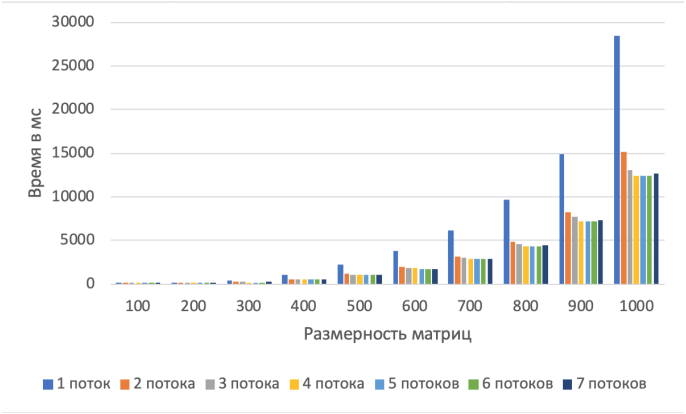

Рисунок 4.1.3. – график зависимости многопоточной версии алгоритма при различном количестве выделенных потоков

Из данных графиков можно сделать вывод, что многопоточная версия алгоритма Винограда работает значительно быстрее чем однопоточная(350% при размерности матриц равной 1000). Многопоточная версия работает одинаково для матриц как с четной, так и с нечетной размерностью. А количество зайдействованных ядер равное 4, 5 и 6 оказалось оптимальным (на 128% быстрее чем на 1 ядре при размерности матриц равной 1000) для утсройства, на котором проводились замеры.

4.2 Вывод

В данном разделе было проведено исследование временных затрат разработанного программного обеспечения, вместе с подробным сравнительным анализом реализованных алгоритмов на основе экспериментальных данных, из которых можно сделать вывод, что многопоточная реализация работает быстрее (на 350% при размерности матриц равной 1000) однопоточной, а также что отимальным количеством ядер являются значения 4, 5

и 6 (на 128% быстрее чем на 1 ядре при размерности матриц равной 1000).

Заключение

В ходе выполнения данной лабораторной работы были изучены и реализованы алгоритмы сортировки. Алгоритмы были разработаны и реализацованы, было проведено исследование временных затрат алгоритмов, а также дано описание и обоснование полученных результатов.

В аналитическом разделе было дано описание стандартного алгоритма и алгоритма Винограда. В конструкторском разделе был формализован и описан процесс вычисления пороизвдения двух матриц, разработаны алгоритмы. В технологическом разделе были рассмотрены требования к разрабатываемому программному обеспечению, средства, использованные в процессе разработки для реализации поставленных задач, а также представлен листинг кода программы. В экспериментальном разеде было проведено исследование временных затрат.

Список литературы

[1] Кибернетический сборник. Новая серия. Вып. 25. Сб. статей 1983 — 1985 гг.: Пер. с англ. — М.: Мир, 1988 — В.Б. Алекссев. Сложность умножения матриц. Обзор.

[2] Энтони Уильямс Параллельное программирование на C++ в действии. Практика разработки многопоточных программ.

[3] ISO/IEC JTC1 SC22 WG21 N 3690 «Programming Languages — C++» [Электронный ресурс]. – Режим доступа: https://devdocs.io/cpp/, свободный. (Дата обращения: 29.09.2019 г.)

[4] Apple «Apple Developer Documentation» [Электронный ресурс]. – Режим доступа: https://developer.apple.com/documentation/, свободный. (Дата обращения: 29.09.2019 г.)