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

Дисциплина: Анализ алгоритмов

Лабораторная работа № 4

ТЕМА
Исследование работы алгоритма Винограда для умножения матриц реализованного при помощи параллельных вычислений

Студент группы ИУ7-54,
Лозовский Алексей

2019 г.

Содержание

Введение

Любая программа может быть оптимизирована различными способами. При решении задач на компьютере нужно учитывать несколько сторон этого вопроса - компьютерную и предметную. Существует множество способов оптимизации, один из них - распараллеливание вычислений, что позволяет одновременно вычислять несколько результатов.

- В математике для компактной записи систем линейных алгебраических или дифференциальных уравнений, как это подробно описано в [1];
- при построении экономических моделей, подробнее см. в [2]

1 Аналитическая часть

1.1 Описание алгоритмов

Согласно [3] и [4] **умножение матриц** — одна из основных операций над матрицами. Матрица, получаемая в результате операции умножения, называется произведением матриц.

Пусть даны две прямоугольные матрицы A, 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 $l \times n$:

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

вычисляется по формуле (1):

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

Для увеличения скорости выполнения умножения матриц был разработан алгоритм Винограда, согласно [5] идея алгоритма заключается в том, чтобы заранее вычислять некоторые значения для дальнейшего использования впоследствии.

Рассмотрим два вектора $V = (v1, v2, v3, v4)$ и $W = (w1, w2, w3, w4)$. Их скалярное произведение равно (2):

$$V * W = v1w1 + v2w2 + v3w3 + v4w4 \tag{2}$$

Выражение (2) можно переписать, как (3):

$$V * W = (v1 + w2)(v2 + w1) + (v3 + w4)(v4 + w3) - v1v2 - v3v4 - w1w2 - w3w4 \tag{3}$$

Выражение (3) допускает предварительную обработку $v1v2, v3v4, w1w2, w3w4$. Их можно вычислить заранее и запомнить для каждой строки первой матрицы и для каждого столбца второй.

Алгоритм умножения матриц может быть оптимизирован не только математически, но и технически - при помощи использования дополнительных потоков. Если при вычислении матрицы разбивать ее на несколько частей и вычислять параллельно, можно получить прирост времени в несколько раз. При этом вычисление матрицы может быть реализовано разными способами. Допустимо изменять порядок циклов: так, стандартный i,j,k может быть вычислен в любом порядке.

Согласно [6] существует несколько подходов распараллеливания: зелёные потоки и нативные. Первые - это потоки выполнения, управление которыми вместо операционной системы

выполняет виртуальная машина (ВМ). Они эмулируют многопоточную среду, не полагаясь на возможности ОС по реализации легковесных потоков. Стандартные потоки создаются внутри процесса и пользуются общими ресурсами. Это означает, что, например, память и декскрипторы файлов, являются общими для всех потоков процесса. Подобный подход и принято называть native threads.

1.2 Задание на выполнение лабораторной работы

- Оптимизировать алгоритм Винограда при помощи распараллеливания вычислений.
- Исследовать поведение функции при обработке различным числом потоков.

2 Конструкторская часть

2.1 Структура кода и представление данных

Для написания кода был выбран структурный стиль программирования. Матрица представлена структурой, хранящей ее описание(количество строк и столбцов). Работа будет осуществляться с целочисленными значениями типа int.

2.2 Разработка алгоритмов

2.2.1 Оптимизация Алгоритма Винограда

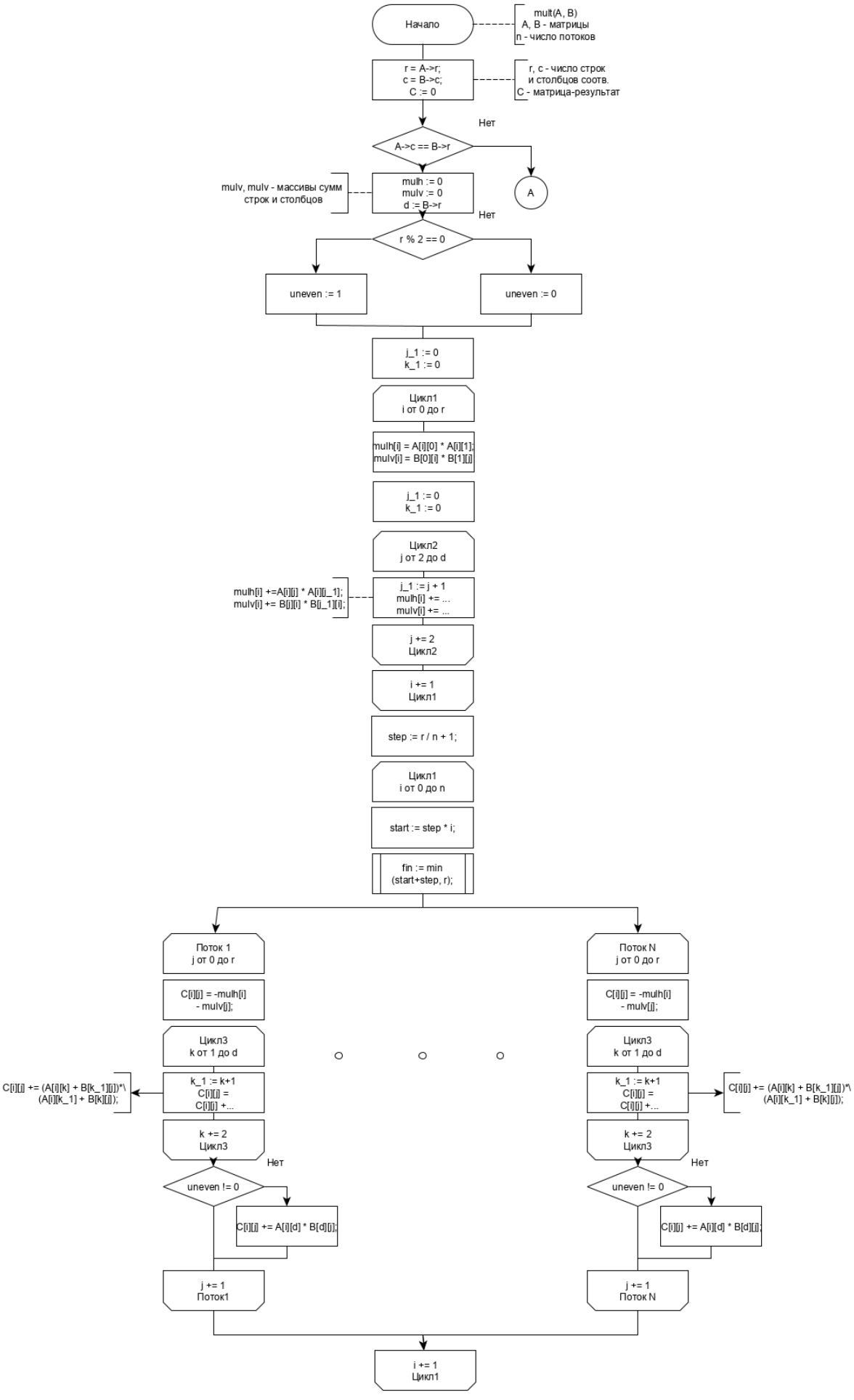


Рис. 1: Алгоритм Винограда с параллельными вычислениями

2.3 Выводы по конструкторскому разделу

Таким образом, был определен стиль кода для реализации алгоритмов, а также разработана схема исследуемого алгоритма.

3 Технологическая часть

3.1 Требования к программному обеспечению

Программа должна поддерживать экспериментальный и пользовательский режимы. В экспериментальном режиме должно производиться тестирование времени работы алгоритма Винограда, реализованного при помощи параллельных вычислений, размерностей $100 \times 100, \ldots, 1000 \times 1000$ и $101 \times 101, \ldots, 1001 \times 1001$ с шагом 100 на диапазоне потоков $2^n, \ldots, n \in [1, +\infty]$. Данные записываются в файл. Границу количества потока определяет программист. Пользовательский режим должен позволять проверку времени обработки для матрицы заданного пользователем размера на заданном количестве потоков, также должна быть предоставлена возможность проверки корректной работы функции.

3.2 Средства реализации

Поскольку программа реализована в структурном стиле, из языков, поддерживающих его был выбран C++, стандарта 2011 года[7], так как на нем имеется наибольший опыт работы. Для реализации потоков была выбрана библиотека <thread> с целью ознакомления с ее возможностями. Для замера времени будем использовать библиотеку <chrono>, так как библиотека <ctime> неверно считает время при параллельных вычислениях.

3.3 Листинг кода

Листинг 1: Функции для обработки в потоке

```
void magic(mtx_t *res, mtx_t *A, mtx_t *B, int *mh, int *mv, int st, int end)
{
    int c = B->c;
    int d = B->r-1;
    int k_1;


    for (int i = st; i < end; i++)
        for (int j=0; j<c; j++){
            res->mtx[i][j] = mh[i] + mv[j];


            for (int k=0; k<d; k+=2){
                k_1 = k+1;


                res->mtx[i][j] += (A->mtx[i][k] + B->mtx[k_1][j])*\
                (A->mtx[i][k_1] + B->mtx[k][j]);
            }
        }
    }
```

```
}
```

Листинг 2: Функции для обработки в потоке

```
mtx_t *vinograd(mtx_t *A, mtx_t *B, int n)
{
    mtx_t *res = nullptr;
    int r = A->r;
    int c = B->c;
    int d = B->r-1;

    bool uneven = r % 2 != 0 ? 1 : 0;


    if (check(A->c, B->r))
    {
        int *mulh = new int[r];
        int *mulv = new int[c];


        int j_1;

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

            mulh[i] = 0;
            mulv[i] = 0;

            for (int j=0; j<d; j+=2){

                j_1 = j+1;

                mulh[i] -= A->mtx[i][j] * A->mtx[i][j_1];
                mulv[i] -= B->mtx[j][i] * B->mtx[j_1][i];

                }
        }


        res = generate_mtx(r, c);


        std::thread *thr = new std::thread [n];
        const int step = r / n + 1;
        int start;

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

            start = step * i;

            thr[i] = std::thread(magic, std::ref(res), A, B,\
                                 mulh, mulv, start,\
                                 std::min(start+step, r));
        }
```

```
        for (int i=0; i<n; i++)
            thr[i].join();


        if (uneven)
            for (int i=0; i<r; i++)
                for (int j=0; j<c; j++)
                    res->mtx[i][j] += A->mtx[i][d] * B->mtx[d][j];

        delete []thr;
        delete []mulh;
        delete []mulv;
    }

    return res;
}
```

3.4 Выводы по конструкторскому разделу

Были описаны требования ПО, приведен листинг кода с реализацией фукнции для подачи в поток, а также был выбран язык программирования и бибилотека для работы с потоками и замером времени работы.

4 Экспериментальная часть

4.1 Примеры работы

```
1. User mode;
2. Experinmetal mode;
0. Exit;
Choose number: 1
1. Check time.
2. Check work.
0. Exit.
Input number: 1
Input thread number: 8
Input matrix size(M): 500
size: 500; time: 0.138955
```

```
Input thread number: 2
Input matrix size(M): 2
1. Generate unit matrixes.
2. Set matrixes from console.
Input number: 1
Matrix A:
1 0
0 1
Matrix B:
1 0
0 1
Result matrix:
1 0
0 1
```

4.2 Постановка эксперимента

Исследуем работу алгоритма и сравним время выполнения при обработке матриц размером $100 \times 100, \ldots, 1000 \times 1000$ и $101 \times 101, \ldots, 1001 \times 1001$ с шагом 100 при обработке 1, 2, ... , 256 потоками. При тестировании будем брать усредененное время из 10 тестов. Время будем замерять в секундах. С помощью библиотеки `<chrono>`

4.2.1 Тестирование времени работы функций

В таблицах 1, 2 отображено время работы алгоритма при обработке квадратных матриц четной и нечетной величин 4 потоками.

Матрица 1	Матрица 2	Время работы, с
100	100	0.002792
200	200	0.010173
300	300	0.031540
400	400	0.081531
500	500	0.167363
600	600	0.303342
700	700	0.478347
800	800	0.987720
900	900	1.218632
1000	1000	2.410327

Таблица 1: Тестирование времени на матрицах четного размера

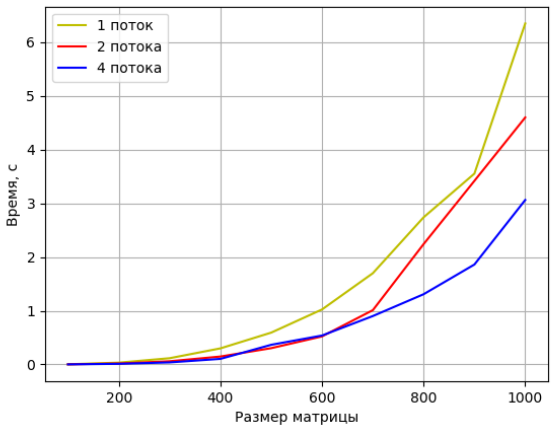
Матрица 1	Матрица 2	Время работы, с
101	101	0.009001
201	201	0.023538
301	301	0.070811
401	401	0.194690
501	501	0.194690
601	601	0.576365
701	701	0.893336
801	801	1.484895
901	901	2.044143
1001	1001	2.959412

Таблица 2: Тестирование времени на матрицах нечетного размера

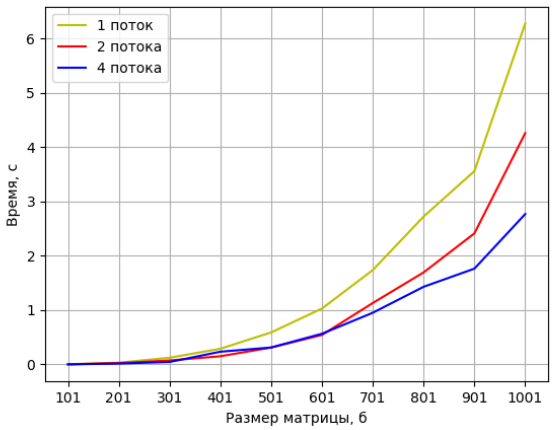
4.3 Сравнительный анализ на материале экспериментальных данных

На рис. 2, 3, 4 отображено время работы алгоритма Винограда для умножения матриц для четных и нечетных размеров соответственно.

На рис.2 показано время работы при 1, 2 , 4 потоках.
На рис. 3 отображено время при 8, 16 потоках.
На рис. 4 при 32, 64 потоках.

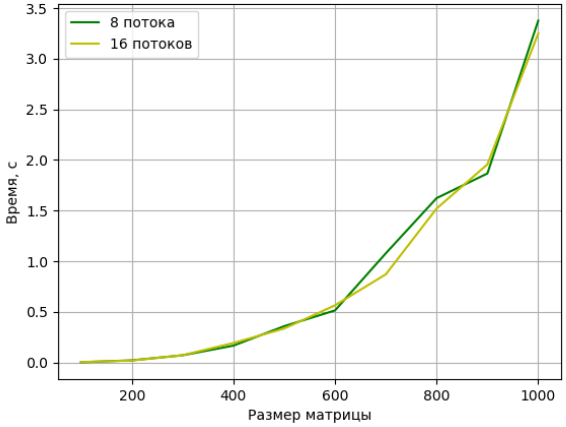


а)

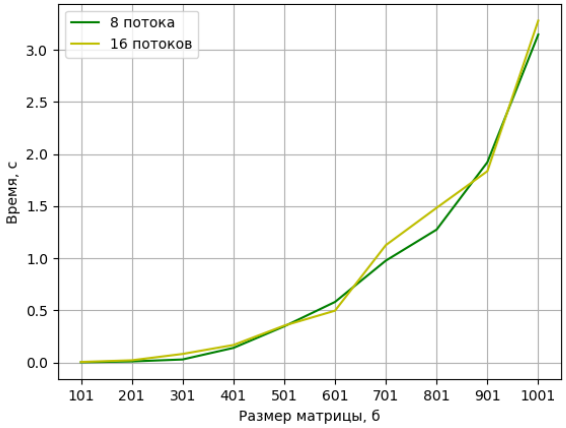


б)

Рис. 2: Сравнение работы алгоритмов при 1, 2, 4 потоках (а) - для четных, (б) для нечетных

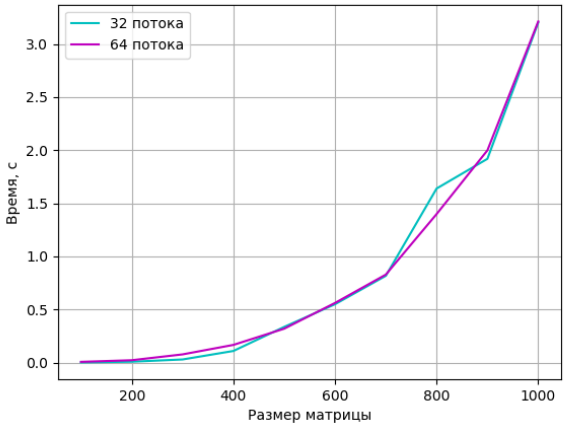


а)

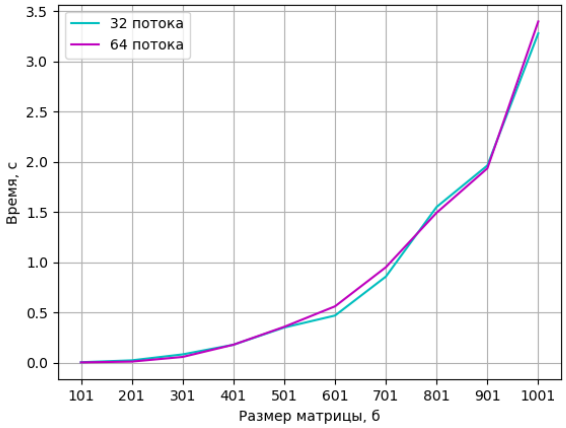


б)

Рис. 3: Сравнение работы алгоритмов при 8 и 16 потоках (а) - для четных, (б) для нечетных



а)



б)

Рис. 4: Сравнение работы алгоритмов при 32 и 64 потоках (а) - для четных, (б) для нечетных

4.4 Выводы по экспериментальной части

На основе полученных в ходе эксперимента данных можно сделать ряд заключений: При данной реализации обработка четных и нечетных матриц занимает примерно одинаковое количество времени, вне зависимости от количества потоков. Лучший результат алгоритм показал при 4 потоках, время работы составило 2.41с для матрицы 1000×1000 и 2.64с для матрицы 1001×1001. Худший результат показал алгоритм при 64 потоках - матрицы тех же размерностей он обработал за 3.396611 и 3.435541 соответственно. Тем не менее, при сравнении времени работы многопоточного вычисления и одногопоточного, первый показал прирост времени почти в 3 раза. При работе с одним потоком было время выполнения составило 6.3508с, при параллельных вычисления результат - 2.64с.

Заключение

Таким образом, в ходе лабораторной работы был оптимизирован и реализован алгоритм Винограда для умножения матриц. Была написана программа, поддерживающая пользовательский и экспериментальный режим, Было проведено тестирование функции умножения для матриц разных размеров на разном количестве потоков $1, \ldots, 64$ и сделаны сделующие выводы: при данной реализации обработка четных и нечетных матриц занимает примерно одинаковое количество времени, вне зависимости от количества потоков. Лучший результат алгоритм показал при 4 потоках, время работы составило 2.41с для матрицы 1000×1000 и 2.64с для матрицы 1001×1001. Худший результат показал алгоритм при 64 потоках - матрицы тех же размерностей он обработал за 3.396611 и 3.435541 соответственно. Тем не менее, при сравнении времени работы многопоточного вычисления и одногопоточного, первый показал прирост скорости почти в 3 раза. При работе с одним потоком было время выполнения составило 6.3508с, при параллельных вычисления результат - 2.64с.

Список литературы

[1] А.Н. Канатников, А.П. Крищенко, "Линейная алгебра и функции нескольких переменных"

[2] Винсковская Л.А., Маркушина А.А. "Способы реализации экономико-математических моделей. Современное развитие России в условиях новой цифровой экономики материалы II Международной научнопрактической конференции"

[3] В. Н. Задорожный, В. Ф. Зальмеж, А.Ю. Трифонов, А. В. Шаповалов, "Высшая математика для технических университетов. Линейная алгебра"

[4] Е. Е. Тыртышников "Матричный анализ и линейная алгебра"

[5] Jeffrey J. McConnell, "Analysis of Algorithms: An Active Learning Approach"

[6] Уильямс Э., "Параллельное программирование на C++ в действии. Практика разработки многопоточных программ"

[7] Стандарт языка C++11 согласно ISO. [Электронный ресурс]. URL: https://isocpp.org/std/the-standard