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

Дисциплина: анализ алгоритмов

Лабораторная работа №4

Студент: Власова Екатерина, гр. ИУ7-54

2019 г.

Содержание

Введение

Многое количество программ, используемых во многих сферах жизни, требует больших вычислительных затрат. Но выполнение таких программ может занимать от нескольких часов до нескольких дней, если использовать только последовательное выполнение действий. На самом деле, во многих задачах некоторые действия можно, хотя бы частично, выполнять параллельно. Многопоточность позволяет приложению параллельно работать над несколькими относительно независимыми задачами и использовать преимущества многопроцессорных систем для повышения производительности приложения. Очевидно, что в любом случае задача, над которой работает приложение, должна обладать одним существенным свойством: ее можно разделить на несколько более или менее независимых подзадач, над которыми можно работать параллельно. В противном случае использование многопоточности просто невозможно. Нередко такое распараллеливание само по себе составляет сложную алгоритмическую задачу, не всегда оно в принципе возможно. Но часто оно возникает естественным образом. В данной лабораторной работе многопоточность рассматривается на примере алгоритма Винограда – умножения матриц.

1 Аналитическая часть

Согласно [9], многопоточность – это свойство платформы или программы, позволяющее процессу состоять из нескольких потоков команд, исполняющихся параллельно, без предписанного порядка во времени. Многопоточность позволяет повысить производительность процесса за счет распараллеливания процессорных вычислений.

Для каждого выполняемого проекта приложения операционная система создает процесс. Согласно [10], процесс - это исполнение программы. Операционная система использует процессы для разделения исполняемых приложений. Поток - это основная единица, которой операционная система выделяет время процессора.

Существуют два вида реализации многопоточности: зелёные потоки и нативные потоки. Зелёные потоки — это потоки, управление которыми вместо операционной системы выполняет виртуальная машина. Эти потоки реализуются на уровне приложений и управляются в пространстве пользователя. Только один зеленый поток может быть обработан за один раз. Следовательно, эта модель считается моделью «многие к одному». Из-за этого зеленые потоки могут работать на многоядерных процессорах, но не могут использовать преимущества нескольких ядер.

Синхронизация и совместное использование ресурсов проще для зеленых потоков и, следовательно, время выполнения также меньше.

Ссылаясь на [16], нативные потоки используют встроенную способность операционной системы управления многопоточными процессами. Несколько нативных потоков могут сосуществовать, поэтому их также называют моделью «многие ко многим». Это позволяет ей в полной мере использовать преимущества многоядерных процессоров и одновременно выполнять потоки на отдельных ядрах. Поскольку эта модель позволяет выполнять несколько потоков одновременно, синхронизация потоков и совместное использование ресурсов становятся сложными. Это увеличивает время выполнения потоков.

1.1 Описание алгоритмов

Ссылаясь на [1], дадим определение матрицы:

Матрицей A размера m×n называется прямоугольная таблица чисел, функций или алгебраических выражений, содержащая m строк и n столбцов. Числа m и n определяют размер матрицы. Условимся обозначать матрицы прописными буквами латинского алфавита: A, B, C, D, … Числа, функции или алгебраические выражения, образующие матрицу, называются матричными элементами. Будем обозначать их строчными буквами с двумя индексами.

Первый индекс i=1,2, …, m указывает номер строки, а второй индекс j=1,2, …, n — номер столбца, в которых располагается соответствующий элемент. Таким образом,

$$A_{m\times n}\begin{pmatrix} a_{11} & a_{21} & .. & a_{1n} \\ a_{21} & a_{22} & .. & a_{2n} \\ .. & .. & .. & .. \\ a_{m1} & a_{m2} & .. & a_{mn} \end{pmatrix} \quad (1)$$

Здесь и в последующих формулах рядом с символом матрицы указан её размер.

Умножение матрицы A на матрицу B определено, лишь когда число столбцов первой матрицы в произведении равно числу строк второй.

1.1.1 Базовый алгоритм умножения матриц

Пусть даны две матрицы A и B. Согласно [1], умножение матрицы A на матрицу B определено, лишь когда число столбцов первой матрицы в произведении равно числу строк второй. Тогда произведением матриц A m×k B k×n называется матрица C m×n, каждый элемент которой cij равен сумме попарных произведений элементов i–й строки матрицы A на соответствующие элементы j–го столбца матрицы B, т. е.

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{ik}b_{jk} = \sum_{s=1}^{k} a_{is}b_{sj} \quad (1.1)$$

для всех i = 1, 2, …, m и j = 1, 2, …, n.

1.1.2 Алгоритм Винограда

Если рассмотреть результат умножения двух матриц, то видно, что каждый элемент в нём представляет собой скалярное произведение соответствующих строки и столбца исходных матриц. Можно заметить также, что такое умножение допускает предварительную обработку, позволяющую часть работы выполнить заранее.

Используя [4], рассмотрим два вектора V = (v1, v2, v3, v4) и W = (w1, w2, w3, w4). Их скалярное произведение равно:

$$V \cdot W = v1w1 + v2w2 + v3w3 + v4w4. \quad (1.2)$$

Это равенство можно переписать в виде:

$$V \cdot W = (v1 + w2)(v2 + w1) + (v3 + w4)(v4 + w3) - v1v2 - v3v4 - w1w2 - w3w4 \quad (1.3)$$

Выражение в правой части формулы 1.3 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.

1.1.3 Параллельный алгоритм Винограда

Чтобы увеличить эффективность алгоритма Винограда, надо распараллелить его часть, содержащую тройной цикл. Результат умножения для каждой строки не зависит от умножения других строк. Поэтому эта часть алгоритма может быть распараллелена. Каждый поток будет выполнять вычисления для определённых строк результирующей матрицы.

1.2 Задание на выполнение лабораторной работы

- Реализовать обычный алгоритм Винограда;
- Распараллелить алгоритм Винограда;
- Оценить трудоёмкость рассматриваемых алгоритмов;
- Сравнить эффективность алгоритмов по времени.
- Получение навыка параллельной реализации алгоритма с целью снижения трудоёмкости выполнения в сравнении с последовательным алгоритмом на примере решения задачи умножения матриц;
- Экспериментальное подтверждение оценок трудоёмкости.

1.3 Вывод по аналитической части

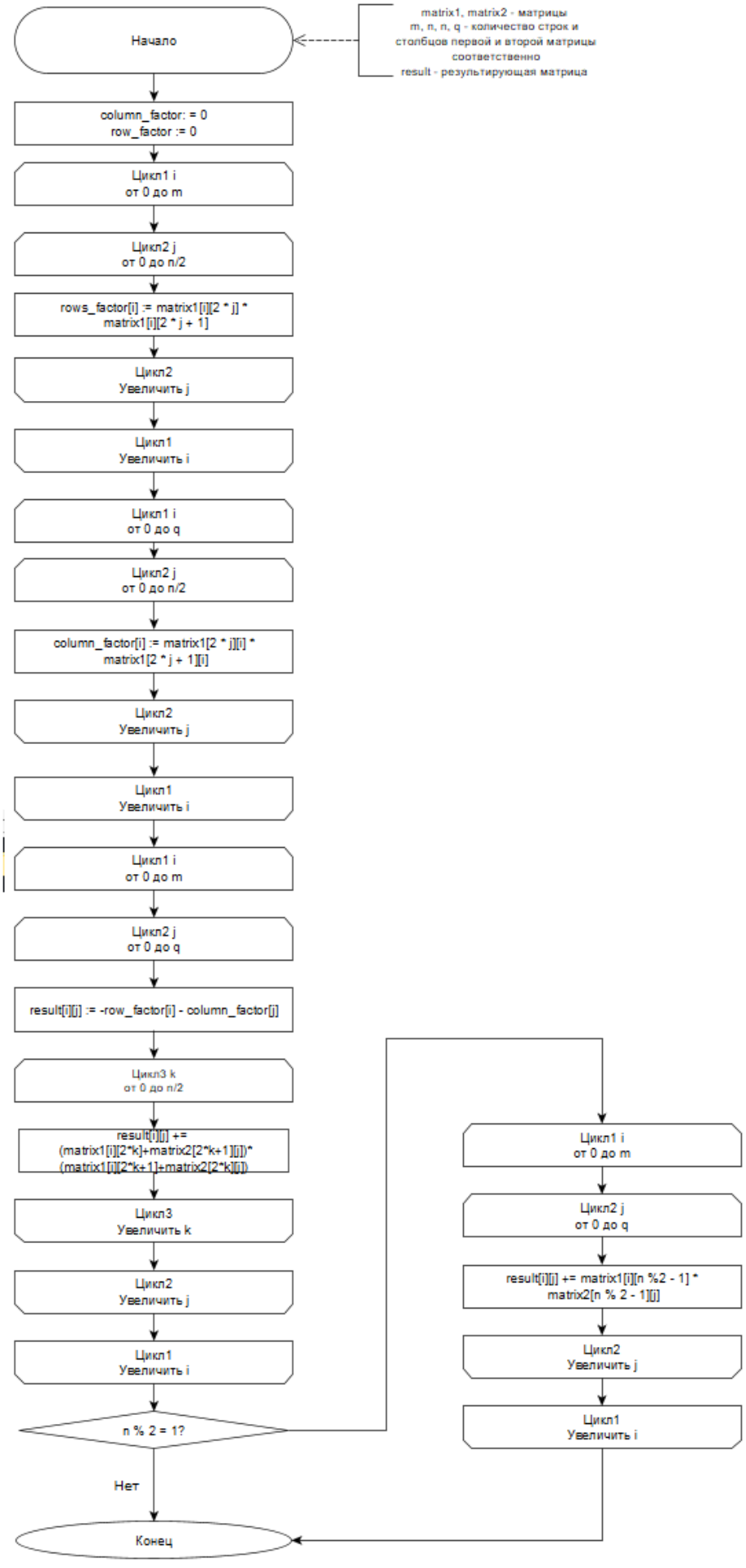
Были описаны классический алгоритм умножения матриц и алгоритм Винограда. Рассмотрен способ распараллелить алгоритм Винограда.

2 Конструкторская часть

В данном разделе представлены блок-схемы рассматриваемых алгоритмов.

2.1 Разработка алгоритмов

2.1.1 Алгоритм Винограда



Fall=2+Frf+ Fcf+Fmult

Frf=2+M(1+ Frf_inner)

Frt_inner = 2 + D(5+ 3 +1 +1

+1 +

+ 1)

Fcf=2+Q(1+ Fcf_inner)

Fcf_inner=2+ D*(5 + 3 +1 +1

+1+

+1)

Fmult=2+M(1 +Fmouter)+ Fodd

Fmouter=2+Q (1+4+1 +1 + Fminner)

Fminner=2 +D(10+5+1+2

+1 +1 +1)

Fodd=4 +(Foddout)

Foddout=2+M (1+Foddiner)

Foddinner=1+ Q(6+1+2+1+2

+1 +1)

Оператор	Количество
Fall	
=	2
/	2
Frf	
<	1
=	1
Frf_inner	
[]	5
+	1
*	3
<	1
=	1
Fcf	
<	1
=	1
Fcf_inner	
*	3
[]	5
=	1
+	1
<	1
Fmult	
<	1
+=	1
Fmouter	
<	1
+=	2
[]	4
*	1
Fminner	
<	1
+=	1
[]	10
*	5
+	3
Fodd	
=	11
%	1
=	1
<	1

Foddout	
=	1
[]	6
%	2
-	1

Рассчитаем трудоёмкость алгоритма Винограда:

$$Foddinner = 1+14Q$$

$$Foddout = 1 + 2M + 14QM$$

$$Fodd = 5 + 2M + 14QM$$

$$Fminner = 2 + 21D$$

$$Fmouter = 2+9Q+21DQ$$

$$Fmult = 2+ 3\ M+9QM+21DQM+5 + 2M + 14QM=7+5M+23QM+21DQM$$

$$Fcfinner = 2 + 12D$$

$$Fcf = 2+3Q+12DQ$$

$$Frfinner = 2+12D$$

$$Frf = 2+3m+12DM$$

$$Fall = 2 +7+5M+23QM+21DQM+2+3Q+12DQ+2+3M+12DM=$$

$$13+8M+23QM+21DQM+3Q+12DQ+12DM$$

2.1.2 Параллельный алгоритм Винограда

2.2 Выводы по конструкторскому разделу

Были сделаны блок-схемы реализуемых алгоритмов. Для каждого алгоритма оценена сложность.

3 Технологическая часть

В данном разделе был сделан выбор используемого языка программирования, представлен код алгоритмов.

3.1 Требования к программному обеспечению

Программа должна поддерживать пользовательский режим, в котором пользователь может с клавиатуры задать размер матриц и значения их элементов; а также операционный режим, в котором пользователь вводит размер матриц, он автоматически заполняется, и для каждого алгоритма выводится время его работы. Во всех режимах пользователь может ввести количество потоков для параллельного алгоритма Винограда.

3.2 Средства реализации

В качестве языка программирования в данной работе был выбран C++ 11 с целью изучения темы многопоточности на языках высоко уровня. Его стандарт представлен в [8].

Для работы с потоками используется библиотека std::thread. Это нативные потоки.[15]

3.3 Листинг кода

3.3.1 Алгоритм Винограда

Листинг 0

```
int** winograd(Matrix& matr1, Matrix& matr2, int** data)
{
	size_t *res = nullptr;
	int r = matr1.rows;
	int c = matr2.columns;
	int* mulh = new int[r];
	for (int i = 0; i < r; i++) {
		mulh[i] = matr1.data[i][0] * matr1.data[i][1];
		for (int j = 2; j <= matr2.rows / 2; j++)
		{
			mulh[i] = mulh[i] + matr1.data[i][2 * j - 2] * matr1.data[i][2 * j - 1];
		}
	}
	int* mulv = new int[c];
	for (int i = 0; i < c; i++) {
		mulv[i] = matr2.data[0][i] * matr2.data[1][i];
		for (int j = 2; j <= matr2.rows / 2; j++)
			mulv[i] = mulv[i] + matr2.data[2 * j - 2][i] * matr2.data[2 * j - 1][i];
	}
	for (int i = 0; i < r; i++)
		for (int j = 0; j < c; j++) {
			data[i][j] = -mulh[i] - mulv[j];
			for (int k = 1; k <= matr2.rows / 2; k++)
				data[i][j] = data[i][j] + \
				(matr1.data[i][2 * k - 2] + matr2.data[2 * k - 1][j]) *
(matr1.data[i][2 * k - 1] + \
					matr2.data[2 * k - 2][j]);
		}
	if (r % 2 != 0)
	{
		for (int i = 0; i < r; i++)
			for (int j = 0; j < c; j++)
				data[i][j] = data[i][j] + matr1.data[i][matr2.rows - 1] *
				matr2.data[matr2.rows - 1][j];
	}
	delete[]mulv;
	delete[]mulh;
	return data;
}
```

3.3.2 Распараллеленный алгоритм Винограда

Листинг 1

```
void paral(Matrix& matr1, Matrix& matr2, int **data, int *mulh, int *mulv, int num, int
NThread)
{
	for (int i = num; i < matr1.rows; i += NThread)
		for (int j = 0; j < matr2.columns; j++) {
			data[i][j] = -mulh[i] - mulv[j];
			for (int k = 1; k <= matr2.rows / 2; k++)
				data[i][j] = data[i][j] + \
				(matr1.data[i][2 * k - 2] + matr2.data[2 * k - 1][j]) *
(matr1.data[i][2 * k - 1] + \
					matr2.data[2 * k - 2][j]);
		}
}

int** parall_winograd(Matrix& matr1, Matrix& matr2, int**data, int NThreads)
{
	size_t* res = nullptr;
	int r = matr1.rows;
	int c = matr2.columns;
	int* mulh = new int[r];
	for (int i = 0; i < r; i++) {
		mulh[i] = matr1.data[i][0] * matr1.data[i][1];
		for (int j = 2; j <= matr2.rows / 2; j++)
		{
			mulh[i] = mulh[i] + matr1.data[i][2 * j - 2] * matr1.data[i][2 * j - 1];
		}
	}
	int* mulv = new int[c];
	for (int i = 0; i < c; i++) {
		mulv[i] = matr2.data[0][i] * matr2.data[1][i];
		for (int j = 2; j <= matr2.rows / 2; j++)
			mulv[i] = mulv[i] + matr2.data[2 * j - 2][i] * matr2.data[2 * j - 1][i];
	}

	thread* t = new thread[NThreads];
	for (int i = 0; i < NThreads; i++)
		t[i] = thread(&paral, ref(matr1), ref(matr2), data, &mulh[0], &mulv[0], i,
NThreads);
	for (int i = 0; i < NThreads; i++)
	{
		if (t[i].joinable())
			t[i].join();
	}
	if (r % 2 != 0)
	{
		for (int i = 0; i < r; i++)
			for (int j = 0; j < c; j++)
				data[i][j] = data[i][j] + matr1.data[i][matr2.rows - 1] *
				matr2.data[matr2.rows - 1][j];
	}
	delete[]mulv;
	delete[]mulh;
	return data;
}
```

3.4 Выводы по технологическому разделу

Был выбран язык программирования C++, определена использующаяся библиотека для замера времени, а также реализованы функции умножения матриц. Описаны требования к ПО.

4 Экспериментальная часть

В данном разделе предоставлены примеры работы программы и тесты для функций.

4.1 Примеры работы

```
Choose mode:
1.User mode.
2.Experimental mode.
Number: 1
Input rows of matrix:2
Input columns of matrix:2
Input elenemt:1
Input elenemt:2
Input elenemt:3
Input elenemt:4
Input rows of matrix:2
Input columns of matrix:2
Input elenemt:1
Input elenemt:2
Input elenemt:3
Input elenemt:4
First matrix:
1 2
3 4
Second matrix:
1 2
3 4
Number of threads:2
Winograd:
7 10
15 22

Parall. Winograd:
7 10
15 22
```

Рисунок 4.1 Пример работы программы

```
Choose mode:
1.User mode.
2.Experimental mode.
Number: 2
Size of matrix:500

Number of threads:2
The time for Winograd: 575 ms
The time for parall. Winograd: 293 ms
```

Рисунок 4.2 Пример работы программы

4.2 Тестирование программы

Матрица 1	Матрица 2	Алгоритм Винограда	Параллельный алгоритм Винограда	Ожидаемый Результат
1	1	1	1	1
1 0 0 0 1 0 0 0 1	1 0 0 0 1 0 0 0 1	1 0 0 0 1 0 0 0 1	1 0 0 0 1 0 0 0 1	1 0 0 0 1 0 0 0 1
1 2 3 4	5 6 7 8	19 22 43 50	19 22 43 50	19 22 43 50
1 2 3 4 5 6 7 8 1 2 3 4 5 6 7 8	1 2 3 4 5 6 7 8 1 2 3 4 5 6 7 8	34 44 54 64 82 108 134 160 34 44 54 64 82 108 134 160	34 44 54 64 82 108 134 160 34 44 54 64 82 108 134 160	34 44 54 64 82 108 134 160 34 44 54 64 82 108 134 160

4.3 Постановка эксперимента

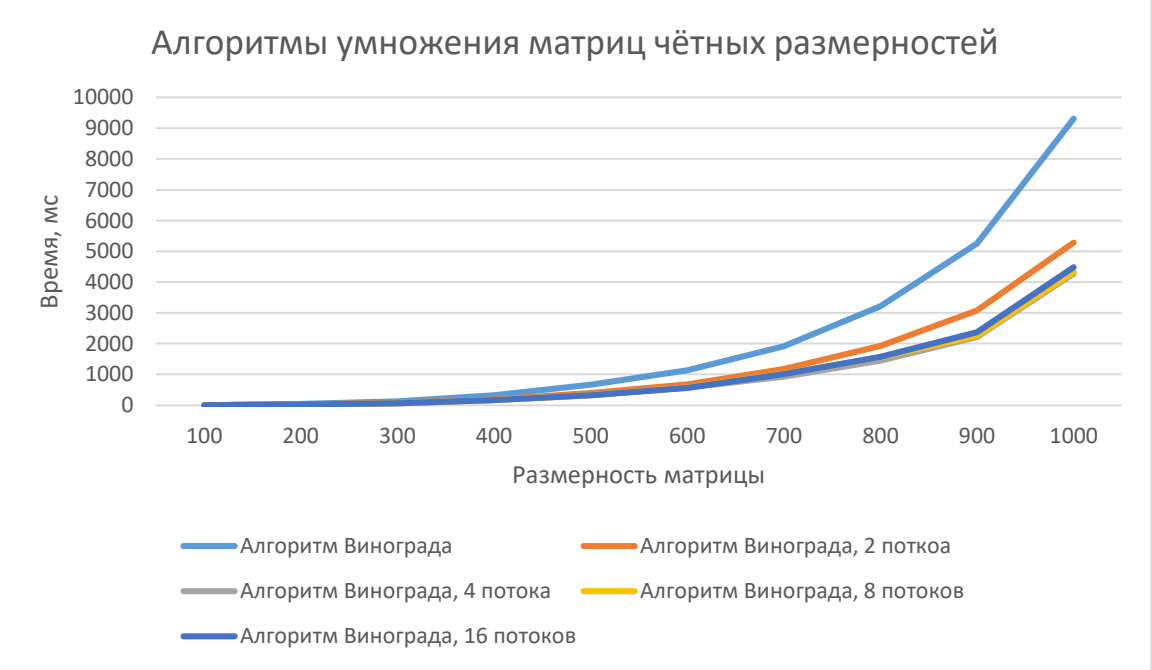
Оценим работу алгоритмов по времени для матриц размерами 100, …, 1000 с шагом 100 и для матриц размерностью 101, …, 1001 с шагом 100. Значения элементов матриц будут выбираться случайно. Параллельный алгоритм проверен на числе потоков, равных 2, 4, 8 и 16. Для получения наиболее точных результатов за время работы функции будем брать среднее значение из 5. Время работы функций засекается с помощью библиотеки chrono. Замеры проводились на Intel Core i5-5200U, ядер: 2, логический процессоров – 4.

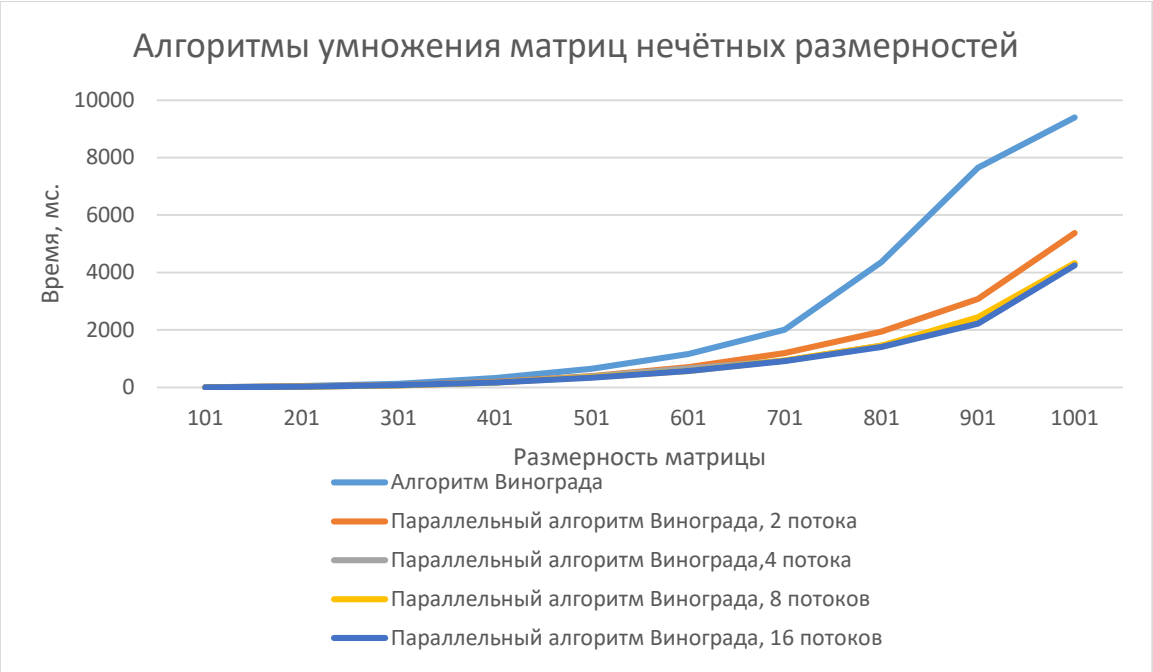
4.3.1 Тестирование времени работы функций

Размерность матрицы	Алг. винограда, мс.	Парал. алг. Винограда (2 потока), мс.	Парал. Алг. Винограда (4 потока), мс.	Парал. Алг. Винограда (8 потоков), мс.	Парал. алг. Винограда(16 потоков), мс
100	5	3	4	5	5
200	42	22	21	21	22
300	129	83	68	69	69
400	320	186	164	168	166
500	664	393	333	327	329
600	1138	677	571	565	560
700	1917	1174	926	1017	1013
800	3212	1929	1444	1563	1577
900	5249	3085	2219	2261	2368
1000	9309	5281	4273	4323	4480

Размерность матрицы	Алг. винограда, мс.	Парал. Алг. Винограда (2 потока), мс.	Парал. алг. Винограда (4 потока), мс.	Парал. алг. Винограда (8 потоков), мс.	Парал. Алг. Винограда(16 потоков)
101	5	5	4	5	5
201	40	29	21	24	23
301	127	82	71	70	80
401	318	189	168	166	166
501	650	394	384	356	329
601	1159	704	662	574	569
701	2012	1196	929	942	917
801	3356	1941	1451	1454	1402
901	7648	3081	2246	2442	2224
1001	9402	5369	4299	4328	4352

4.4 Сравнительный анализ на основе экспериментальных данных





4.5 Выводы

В результате эксперимента получено, что алгоритм Винограда и распараллеленные алгоритмы Винограда считают за приблизительно одинаковое время на матрицах малых размерах (до 200 элементов). Для матрица чётной размерности 200 обычный алгоритм Винограда считает за 42 мс., параллельный алгоритм Винограда на 2 потоках – 21 мс., на 4 -21 мс., на 8 – 21 мс., а на 16 – 22 мс.. Для матрицы нечётной размерности 201 результат приблизительно такой же – обычный алгоритм Винограда за 42 мс., параллельный алгоритм Винограда на 2 потоках – за 29 мс., на 4 – 21, на 8 - 24, 23. Чем больше размерность матрицы, тем больше времени занимают алгоритмы. Лучший результат по времени показывает распараллеленный алгоритм Винограда на 4 потоках. Так, на матрице размерностью 1000 на 1000, распараллеленный алгоритм Винограда на 4 потока – за 4273 мс., а обычный алгоритм Винограда показывает время в 9309 мс., параллельный алгоритм Винограда на 2 потоках – 5281 мс., на 8 – 4323, а 16 - 4480. На матрице размерностью 1001 параллельный алгоритм Винограда на 4 потоках также показывает лучший результат – 4299 мс., на двух потоках – 5369 мс., на 8 – 4328 мс., на 16 – 4352 мс. А обычный алгоритм Винограда – за 9402 мс.

Заключение

В ходе выполнения данной лабораторной работы были реализованы алгоритм Винограда и распараллеленный алгоритм Винограда. Был получен навык параллельный реализации алгоритмов. Оценена трудоёмкость рассматриваемых алгоритмов, реализовано экспериментально её подтверждение.

Сравнена временная эффективность алгоритмов. На матрицах малых размерностей (меньше 300 элементов) алгоритм Винограда и распараллеленный алгоритм Винограда для всех потоков работают приблизительно одинаково. Для матрица чётной размерности 200 обычный алгоритм Винограда считает за 42 мс., параллельный алгоритм Винограда на 2 потоках – 21 мс., на 4 -21 мс., на 8 – 21 мс., а на 16 – 22 мс.. Для матрицы нечётной размерности 201 результат приблизительно такой же – обычный алгоритм Винограда за 42 мс., параллельный алгоритм Винограда на 2 потоках – за 29 мс., на 4 – 21, на 8 - 24, 23. Чем больше размерность матрицы, тем больше времени занимают алгоритмы. Лучший результат по времени показывает распараллеленный алгоритм Винограда на 4 потоках. Так, на матрице размерностью 1000 на 1000, распараллеленный алгоритм Винограда на 4 потока – за 4273 мс., а обычный алгоритм Винограда показывает время в 9309 мс., параллельный алгоритм Винограда на 2 потоках – 5281 мс., на 8 – 4323, а 16 - 4480. На матрице размерностью 1001 параллельный алгоритм Винограда на 4 потоках также показывает лучший результат – 4299 мс., на двух потоках – 5369 мс., на 8 – 4328 мс., на 16 – 4352 мс. А обычный алгоритм Винограда – за 9402 мс.

Список использованной литературы

1. Белоусов И. В. Матрицы и определители, 2006 г.
2. Корнг Г., Корн Т., Справочник по математике, 1973 г.
3. Хорн Р., Джонсон Ч., Матричный анализ, 2013 г.
4. Беллман Р. Введение в теорию матриц, 1969.
5. Голуб Дж., Ван Лоун Ч., Матричные вычисления, 1999.
6. Ланкастер П., Теория матриц, 1973.
7. Курош А. Г. Курс высшей алгебры, 1968.
8. Pete Becker, Working Draft, Standard for Programming Language C++, 2011.
9. Алексей Куканов, Многопоточность и параллелизм в C++, 2015.
10. Р. Миллер, Л. Боксер Последовательные и параллельные алгоритмы: общий подход, 2006.
11. Multi-threading, [Электронный ресурс] URL: http://www.cplusplus.com/reference/multithreading/
12. Введение в технологии параллельного программирования [Электронный ресурс] URL: https://software.intel.com/ru-ru/articles/writing-parallel-programs-a-multi-language-tutorial-introduction/
13. Технология параллельных вычислений [Электронный ресурс] URL: https://ita.sibsutis.ru/sites/csc.sibsutis.ru/files/courses/pvt/%20параллельных%20вычислений.pdf
14. Библиотека chrono [Электронный ресурс] URL: https://docs.microsoft.com/ru-ru/cpp/standard-library/chrono?view=vs-2017
15. Библиотека std::thread, [Электронный ресурс] URL: https://en.cppreference.com/w/cpp/thread/thread
16. What is the difference between "green" threads and "native" threads? [Электронный ресурс] URL: http://www.jguru.com/faq/view.jsp?EID=143462