Лабораторная работа №2

1. Постановка задачи

Перемножить 2 квадратные матрицы размера 1024x1024 с элементами типа double.

Исходные матрицы генерируются в программе (случайным образом либо по определенной формуле) или считываются из заранее подготовленного файла.

Оценить сложность алгоритма по формуле $c = 2n^3$, где n - размерность матрицы.

Оценить производительность в MFlops, $p = \frac{c}{t} \times 10^{-6}$, где t - время в секундах работы алгоритма.

Выполнить 2 варианта перемножения и их анализ и сравнение:

1-й вариант перемножения – по формуле из линейной алгебры.

2-й вариант перемножения – оптимизированный алгоритм на выбор.

2. Описание решения.

Программа написана на языке C++ и скомпилирована в Visual Studio 2022, настройки проекта Visual Studio и характеристики ПК следующие:

Версия пакета SDK для Windows	**10.0 (последняя установленная версия)**
Набор инструментов платформы	**Visual Studio 2022 (v143)**
Стандарт языка C++	**Стандарт ISO C++14 (/std:c++14)**

Рисунок 1 - настройки проекта

Процессор:	AMD Ryzen 7 3700X 8-Core Processor	3.60 GHz
Установленная память (ОЗУ):	32,0 ГБ 3200 Mhz	

Рисунок 2 - характеристики ПК

Для заполнения матриц создал генератор псевдослучайных целых чисел, но в связи с тем, что в условиях задачи сказано, что элементы должны быть

типа double, при заполнении матриц я добавляю к случайно созданному целому числу 0.3 (первое что пришло в голову), тем самым конвертируя целое число в число с плавающей запятой. Замеры времени производится с помощью библиотеки chrono. Оптимизированный алгоритм перемножения матриц включает в себя транспонирование второй матрицы и перемножение векторов с использованием развёртки цикла в 4 раза.

3. Тесты

Прежде чем считать произведение больших матриц, нужно убедиться, что алгоритм работает правильно. Для этого я создал 2 матрицы 4x4, заполненные с помощью своего генератора псевдослучайных чисел и проверил результат на калькуляторе в интернете.

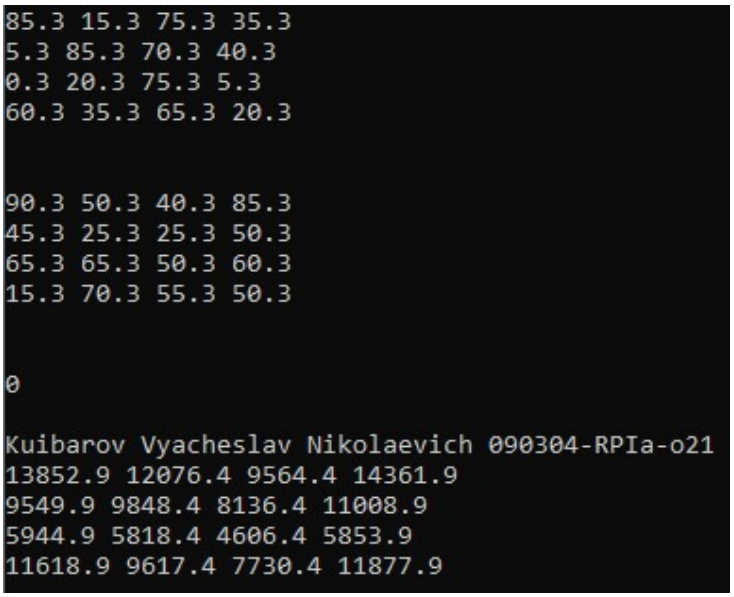
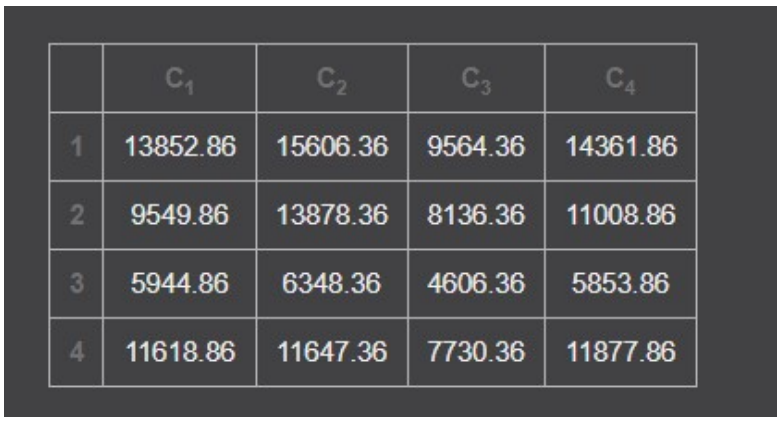


Рисунок 3 - результат программы для матрицы 4x4



	C_1	C_2	C_3	C_4
1	13852.86	15606.36	9564.36	14361.86
2	9549.86	13878.36	8136.36	11008.86
3	5944.86	6348.36	4606.36	5853.86
4	11618.86	11647.36	7730.36	11877.86

Рисунок 4 - результат умножения матриц 4x4, проверенный спомощью онлайн калькулятора

Алгоритм работает корректно, следовательно, можно приступать вычислениям над большими матрицами. Я рассмотрю скорость выполнения программы для матриц не только размерности 1024, но и 2048, 3072 и 4096. Замеры для каждой размерности проводились по 3 раза и находилось среднее арифметическое среди них.

Таблица 1 - время выполнения программы

Размерность	1024	2048	3072	4096
«В лоб»	10.2 сек	54.1	-	-
Оптимизированно	0.3 сек	4.1	13.2	30.4

Как видно, оптимизированный алгоритм больше чем на порядок быстрее чем алгоритм решения «в лоб».

Таблица 2 - сложность алгоритма

Размерность	1024	2048	3072	4096
Сложность	2×2^{30}	2×2^{33}	$1.5^3 \times 2^{34}$	$2 * 2^{36}$

Таблица 3 - Производительность (в MFlops)

Размерность	1024	2048	3072	4096
«В лоб»	210 MFlops	317 MFlops	-	-
Оптимизированно	7158 MFlops	4190 MFlops	4392 MFlops	4521 MFlops

Как видно, оптимизированный алгоритм проявляет себя намного лучше, чем алгоритм перемножения «в лоб», и в плане времени выполнения, и в плане производительности.

4. Листинг кода

```
#include <iostream>
#include <chrono>

using namespace std;

int n = 1024;

// генератор псевдослучайных чисел
unsigned int prevRandom = 5;
unsigned pseudoRandom() {
	prevRandom = prevRandom * 1537 % 1923455;

	return prevRandom % 100;
}

// Заполнение матрицы
void fillMatrix(double* M) {
	for (int i = 0; i < n; i++) {
		for (int j = 0; j < n; j++) {
			M[i * n + j] = pseudoRandom() % 100 + 0.3;
		}
	}
}

// вывод матрицы
void outputMatrix(double* M) {
	for (int i = 0; i < n; i++) {
		for (int j = 0; j < n; j++) {
			printf("%.1lf ", M[i * n + j]);
		}
		cout << endl;
```

```
	}
	cout << endl << endl;
}

// перемножение «в лоб»
void multiply1(double* A, double* B, double* C) {
	int t;
	for (int i = 0; i < n; i++) {
		t = i * n;
		for (int j = 0; j < n; j++) {
			for (int k = 0; k < n; k++) {
				C[t + j] += A[t + k] * B[k * n + j];
			}
		}
	}
}

// создании транспонированной матрицы из данной
double* makeTransponatedMatrix(double* M) {
	double* T = new double[n * n];

	double* rowM;

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

		for (int j = 0; j < n; j++) {
			T[j * n + i] = rowM[j];
		}
	}

	return T;
}
// умножение двух векторов с развёрткой цикла
double multiplyVectors(double* vectorA, double* vectorB) {
	int n2 = (n / 4) * 4;
	double s1 = 0.0, s2 = 0.0, s3 = 0.0, s4 = 0.0;
```

```
	for (int i = 0; i < n2; i += 4) { // loop unroll x4
		s1 += vectorA[i] * vectorB[i];
		s2 += vectorA[i + 1] * vectorB[i + 1];
		s3 += vectorA[i + 2] * vectorB[i + 2];
		s4 += vectorA[i + 3] * vectorB[i + 3];
	}

	for (int i = n2; i < n; i++) {
		s1 += vectorA[i] * vectorB[i];
	}

	return (s4 + s3) + (s2 + s1);
}
// умножение с транспонированием и умножением векторов с развёрткой цикла
void multiply2(double* A, double* B, double* C) {
	double* T = makeTransponatedMatrix(B);

	double* rowA;
	double* rowC;

	for (int i = 0; i < n; i++) {
		rowA = A + i * n;
		rowC = C + i * n;

		for (int j = 0; j < n; j++) {
			rowC[j] = multiplyVectors(rowA, T + j * n);
		}
	}
}

int main() {
	double* A = new double[n * n];
	double* B = new double[n * n];
	double* C = new double[n * n];

	for (int i = 0; i < n * n; i++) {
		C[i] = 0;
	}
```

```
	fillMatrix(A);
	fillMatrix(B);

	// для замеров времени
	typedef chrono::high_resolution_clock Clock;
	auto t1 = Clock::now();

	multiply2(A, B, C);

	auto t2 = Clock::now();
	// Время замеряется в миллисекундах и конвертируется в секунды
	cout << chrono::duration_cast<chrono::milliseconds>(t2 - t1).count() / 1000.0 << endl;

	cout << endl << "Kuibarov Vyacheslav Nikolaevich 090304-RPIa-o21" << endl;

	return 0;
}
```