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»

Лабораторная работа №4
по дисциплине "Анализ Алгоритмов"

Тема Параллельное сложение матриц

Студент Сабуров С. М.

Группа ИУ7-53Б

Преподаватель Волкова Л. Л.

Москва

2021 г.

СОДЕРЖАНИЕ

Введение

Многопоточность — способность центрального процессора (ЦПУ) или одного ядра в многоядерном процессоре одновременно выполнять несколько процессов или потоков, соответствующим образом поддерживаемых операционной системой. Многопоточная парадигма стала более популярной с конца 1990-х годов, поскольку усилия по дальнейшему использованию параллелизма на уровне инструкций застопорились. Смысл многопоточности — квазимногозадачность на уровне одного исполняемого процесса. Значит, все потоки процесса помимо общего адресного пространства имеют и общие дескрипторы файлов. Выполняющийся процесс имеет как минимум один (главный) поток.

Целью данной работы является реализация и анализ алгоритмов параллельного вычисления на примере сложения матриц, .

Для достижения поставленной цели необходимо выполнить следующие задачи:

— изучить понятие параллельных вычислений;
— привести схемы классического и параллельного сложения матриц.
— реализовать классический алгоритм сложения матриц;
— реализовать параллельный алгоритм сложения матриц;
— сравнить их временные характеристики экспериментально;
— на основании проделанной работы сделать выводы.

1 Аналитическая часть

В данном разделе будут рассмотрены алгоритмы сложения матриц

1.1 Стандартный алгоритм

Пусть даны две прямоугольные матрицы

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{l1} & a_{l2} & \cdots & a_{lm} \end{pmatrix}, \tag{1}$$

$$B = \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1m} \\ b_{21} & b_{22} & \cdots & b_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ b_{l1} & b_{l2} & \cdots & b_{lm} \end{pmatrix}. \tag{2}$$

Тогда матрица C размерностью $l \times m$

$$C = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1m} \\ c_{21} & c_{22} & \cdots & c_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ c_{l1} & c_{l2} & \cdots & c_{lm} \end{pmatrix}, \tag{3}$$

в которой:

$$c_{ij} = a_{ij} + b_{ij} \quad (i = \overline{1,l}; j = \overline{1,m}) \tag{4}$$

будет называться сложением матриц A и B. Стандартный алгоритм реализует данную формулу.

1.2 Параллельный алгоритм

Поскольку все элементы матрицы вычисляется независимо друг от друга и матрицы и остаются неизменными, для параллельного вычисления произведения достаточно распределить вычисление элементов матрицы между потоками. Поскольку мы имеем некие аппаратные ограничения, производить данные вычисления для каждого элемента результирующей матрицы в отдельности не эффективно. Следовательно, данная проблема решается разделением элементов результирющей матрицы по строкам и параллельным вычислением результатов для каждого из разделов.

1.3 Вывод

Были рассмотрены алгоритмы классического сложения матриц и параллельного.Поскольку в стандартном алгоритме сложения матриц элементы результирующей матрицы вычисляются независимо друг от друга, есть возможность реализовать параллельный алгоритма.

— Входные данные : количество строк в первой матрице, количество столбцов в первой матрице, элементы первой матрицы, Количество строк во второй матрице, количество столбцов во второй матрице, элементы второй матрицы.
— Выходные данные : на выходе имеем матрицу - результат сложения двух матриц, являющихся входными данными.
— Ограничения, в рамках которых будет работать программа : размеры матриц должны быть целыми положительными числами, элементы матриц должные быть также числами(допустим вещественный тип).
— Функциональные требования : функции, представленные на листингах 2 - 3 должны вычислять результат умножения двух матриц.
— Требования к программному обеспечению : к программе предъявляется ряд требований:
    — на вход подаются размеры матриц (натуральные числа) и самы матрицы, которые нужно сложить;
    — на выходе - результаты сложения матриц алгоритмами простого сложения матриц, параллельного сложения матриц.

2 Конструкторская часть

В данном разделе будут приведены схемы алгоритмов, описание структур данных, способы тестирования и классы эквивалентности.

2.1 Разработка алгоритмов

На рисунках 1-2 приведены схемы алгоритмов простого сложения матриц и параллельного сложения матриц соответственно.

2.2 Описание структур данных

Был реализован класс Matrix, объединивший в себе алгоритмы работы с матрицей и элементы матрицы. Данный класс состоит из массива указателей на строки хранимой матрицы, операции ввода-вывода, а так же метод сложения матриц. Ко всему прочему, в данном классе присутствуют конструкторы и деструктор. На рисунке 3 изображена диаграмма класса Matrix.

2.3 Способы тестирования и классы эквивалентности

Была выбрана методика тестирования черным ящиком. Классы эквивалентности:

— Матрицы одинкаовых размеров.
— Количество столбцов первой матрицы равно количеству строк матрицы, при этом матрицы не одинаковых размеров.
— Матрицы представляют собой 1 элемент.
— Количество столбцов первой матрицы не равно количеству строк матрицы.

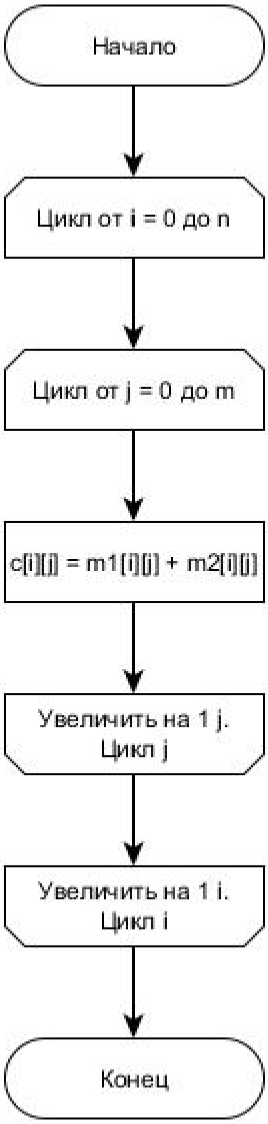

Рис. 1 — Схема алгоритма простого сложения матриц

2.4 Вывод

На основе теоретических данных, полученных из аналитического раздела, были построены схемы требуемых алгоритмов, описаны стртуктуры данных, выделены способы тестирования и классы эквивалентности.

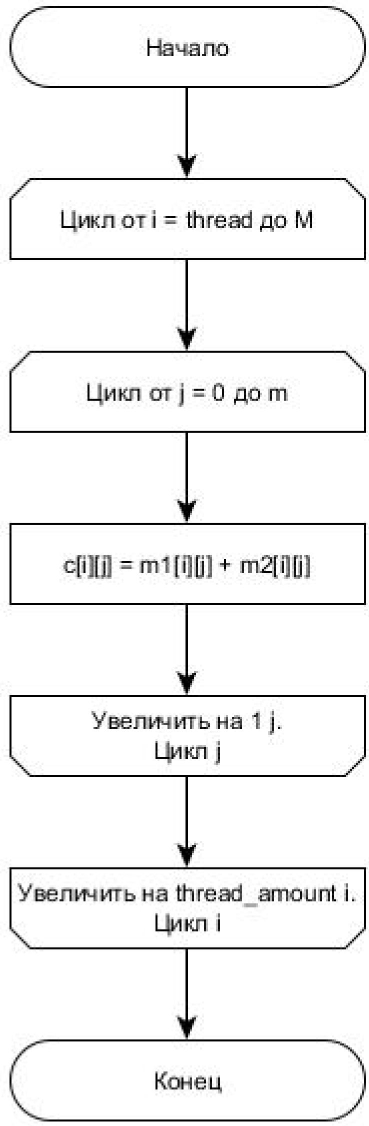

Рис. 2 — Схема алгоритма параллельного сложения матриц

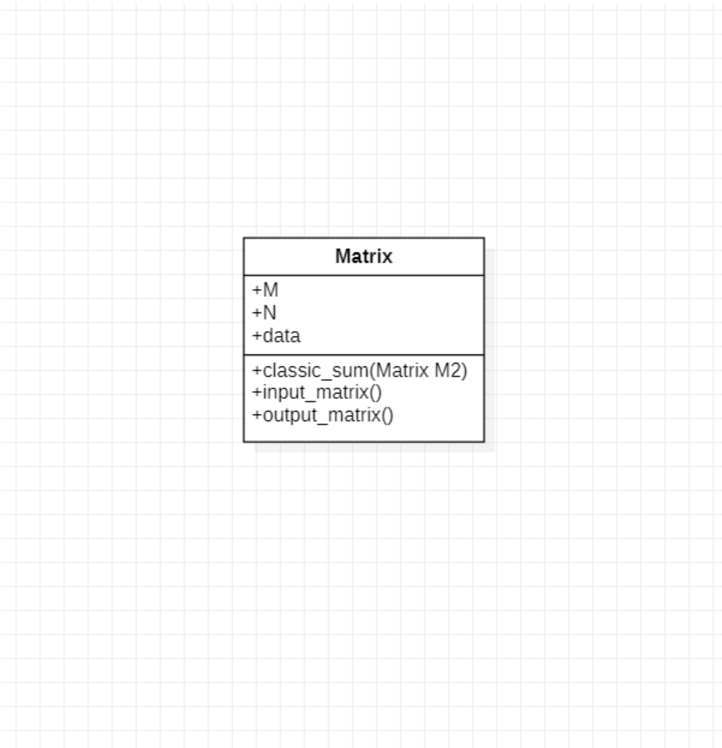

Рис. 3 — Диаграмма класса Matrix.

3 Технологическая часть

В данном разделе приведены требования к программному обеспечению, средства реализации и листинги кода.

3.1 Средства реализации

Для реализации программ был выбран язык программирования C++ [1]. Данный язык был выбран потому, что в нем присутствует инструментарий для замера процессорного времени и тестирования.

3.2 Листинги кода

Листинг 1: Описание класса Matrix

```
template<class T>
class Matrix
{
public:
        int M;
        int N;
        vector<vector<T>> data;
        Matrix();
        Matrix(const Matrix<T>& M_);

        Matrix(int M_, int N_);


        Matrix<T> classic_sum(   Matrix<T>& M2);

        ~Matrix<T>();

        Matrix<T>& input_matrix();

        void output_matrix();

};
```

Листинг 2: Алгоритм простого сложения матриц

```
template <typename T>
Matrix<T> Matrix<T>::classic_sum( Matrix<T>& M2)
{
        Matrix<T> res(this->M, M2.N);
        for (int i = 0; i < res.M; i++)
        {
                for (int j = 0; j < res.N; j++)
                {
                        res.data[i][j] = this->data[i][j] +
   M2.data[i][j];
                }
        }
        return res;

}
```

Листинг 3: Алгоритм параллельного сложения матриц

```
template<class T>
void parallel_add(Matrix<T> matrix1, Matrix<T> matrix2,
   Matrix<T> &result, int thread, int threads_amount)
{

        for (int i = thread; i < result.M; i +=
   threads_amount)
        {
                for (int j = 0; j < result.N; j++)
                {
                        result.data[i] [j] = (matrix1.data[i
   ][j] + matrix2.data[i][j]);
                }

        }
}
template<class T>
Matrix<T>  parallel_sum( Matrix<T> M1,  Matrix<T> M2, int
   threads_amount)
{
        Matrix<T> res(M1.M, M2.N);

```

```
        vector<thread> threads(threads_amount);

        for (int thread_ = 0; thread_ < threads_amount; thread_++)
        {
                threads[thread_] = thread(parallel_add<T>, M1, M2, std::ref(res), thread_, threads_amount);
        }

        for (int i = 0; i < threads_amount; i++)
        {
                threads[i].join();
        }

        return res;
}
```

3.3 Тестирование функций

В таблице 1 приведены модульные тесты для функций сложения матриц выше перечисленными методами. Все тесты были пройдены успешно.

Таблица 1 — Тестирование функций сложения матриц

Матрица 1	Матрица 2	Ожидаемый результат
$\begin{pmatrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\end{pmatrix}$	$\begin{pmatrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\end{pmatrix}$	$\begin{pmatrix}2 & 4 & 6\\8 & 10 & 12\\14 & 16 & 18\end{pmatrix}$
$\begin{pmatrix}1 & 2\\4 & 5\end{pmatrix}$	$\begin{pmatrix}1 & 2\\4 & 5\end{pmatrix}$	$\begin{pmatrix}2 & 4\\8 & 10\end{pmatrix}$
$\begin{pmatrix}8\end{pmatrix}$	$\begin{pmatrix}4\end{pmatrix}$	$\begin{pmatrix}12\end{pmatrix}$
$\begin{pmatrix}1 & 2\end{pmatrix}$	$\begin{pmatrix}3\end{pmatrix}$	Сложение невозможно

3.4 Вывод

Были разработаны и протестированы реализации алгоритмов: простой алгоритм сложения матриц, параллельный алгоритм сложения матриц.

4 Исследовательская часть

В данном разделе будут приведены результаты исследовательской деятельности - замеры процессорного времени работы алгоритмов и тестирование алгоритмов.

4.1 Технические характеристики

Технические характеристики электронно-вычислительнй машины, на которой выполнялось тестирование:

- операционная система: Windows 10 64-bit;
- оперативная память: 8 гигабайт ;
- процессор: Intel i5 7th gen.

Тестирование проводилось на ноутбуке, включенном в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения рабочего стола, окружением рабочего стола, а также непосредственно системой тестирования.

4.2 Время выполнения алгоритмов

Был проведен замер времени работы каждого из алгоритмов с помощью функции std::chrono::system clock::now. Эта функция замеряет процессорное время выполнения функции и усредняет его (проводится 20 замеров). В таблице 2 содержатся результаты исследований.

На рисунках 4, 5 демонстрируется зависимость времени выполнения конкретных реалзиаций алгоритмов сложения матриц от размера стороны квадратной матрицы и количества потоков соответственно.

Таблица 2 — Время выполнения реализаций алгоритмов (в секундах) при количестве потоков 4.

Размер	К	П
100	0.021534	0.020617
200	0.040475	0.039583
500	0.518521	0.197411
1000	2.053641	0.519283

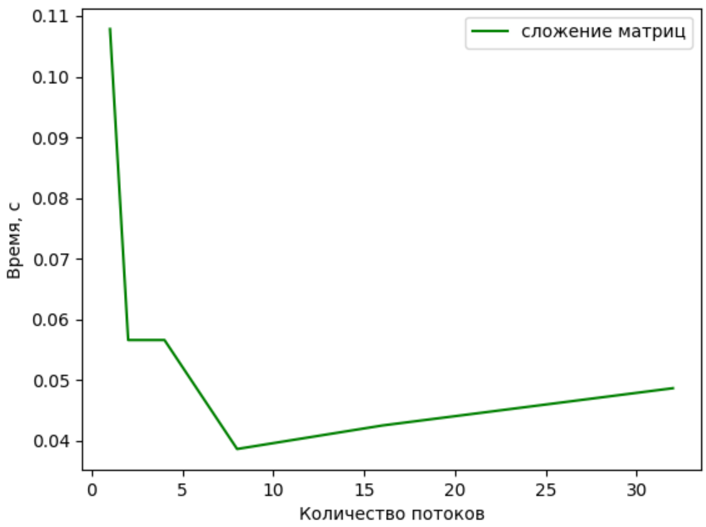

Рис. 4 — Зависимость времени выполнения алгоритмов от количества потоков при размере матрицы 1000 на 1000

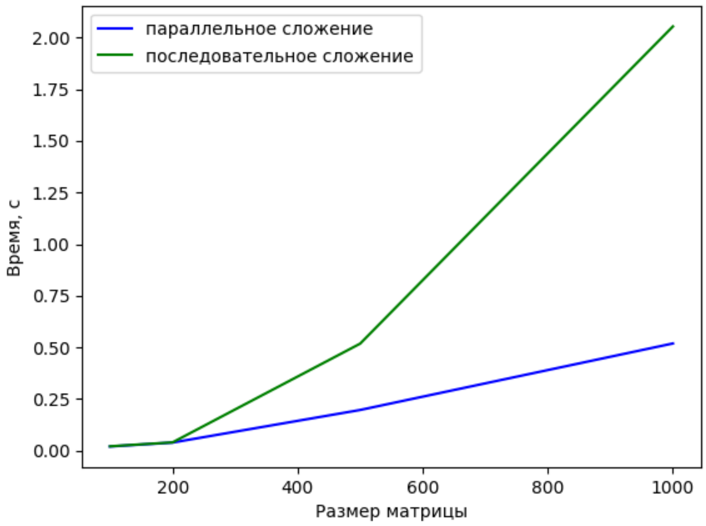

Рис. 5 — Зависимость времени выполнения алгоритмов от размера стороны квадратной матрицы при количестве потоков 4

4.3 Вывод

В данном разделе было произведено сравнение количества затраченного времени вышеизложенных алгоритмов. Наиболее эффективным по времени алгоритмом при работе с матрицами больших размерностей (более 200 элементов) оказался параллельный алгоритм, работащий на 8 потоках. При работе с матрицами малых размерностей (менее 200 элементов) стандартный алгоритм оказался более эффективным (в 4 раза в сравнении с 16 потоками), что связано с дополнительным затратами, которые необходимы при реализации многопоточности (создание потоков, реализация совместного доступа к ресурсам).

Заключение

В ходе выполнения работы была достигнута цель выполнены все поставленные задачи:

— реализовать классический алгоритм сложения матриц;
— реализовать параллельный алгоритм сложения матриц;
— сравнить их временные характеристики экспериментально;
— на основании проделанной работы сделать выводы.

Экспериментально были установлены различия в производительности различных алгоритмов сложения матриц. Параллельный алгоритм имеет большую эффективность(при размерностях выше 200), нежели классический алгоритм сложения матриц.

Список литература

1. Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн. Алгоритмы. Построение и анализ. Издательский дом “Вильямс”, 2011. 823 - 869.
2. Б. Страуструп. Язык программирования C++ . Addison-Wesley, 2000. 142 - 149.
3. Г. Шилдт. C++. Полное руководство. СПб.: Наука и Техника, Издательский дом “Вильямс”, 2006. 621 - 693.
4. Я. Галовиц. C++17 STL. Стандартная библиотека шаблонов. Серийная библиотека программиста, 2018. 91 - 123.
5. Р. Седжвик. Фундаментальные алгоритмы C++. Diasoft, 2001. 42 - 69.