МГТУ им. Баумана

Лабораторная работа №5

По курсу: "Анализ алгоритмов"

Параллельное умножение матриц

Работу выполнил: Ковалев Дмитрий ИУ7-53Б

Преподаватели: Волкова Л.Л., Строганов Ю.В.

Москва, 2020

Оглавление

Введение

Многопоточность — способность центрального процессора (CPU) или одного ядра в многоядерном процессоре одновременно выполнять несколько процессов или потоков, соответствующим образом поддерживаемых операционной системой.

Этот подход отличается от многопроцессорности, так как многопоточность процессов и потоков совместно использует ресурсы одного или нескольких ядер: вычислительных блоков, кэш-памяти ЦПУ или буфера перевода с преобразованием (TLB).

В тех случаях, когда многопроцессорные системы включают в себя несколько полных блоков обработки, многопоточность направлена на максимизацию использования ресурсов одного ядра, используя параллелизм на уровне потоков, а также на уровне инструкций.

Поскольку эти два метода являются взаимодополняющими, их иногда объединяют в системах с несколькими многопоточными ЦП и в ЦП с несколькими многопоточными ядрами.

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.

Многопоточность (как доктрину программирования) не следует путать ни с многозадачностью, ни с многопроцессорностью, несмотря на то, что операционные системы, реализующие многозадачность, как правило, реализуют и многопоточность.

Достоинства:

- облегчение программы посредством использования общего адресного пространства;
- меньшие затраты на создание потока в сравнении с процессами;
- повышение производительности процесса за счёт распараллеливания процессорных вычислений;
- если поток часто теряет кэш, другие потоки могут продолжать использовать неиспользованные вычислительные ресурсы.

Недостатки:

- несколько потоков могут вмешиваться друг в друга при совместном использовании аппаратных ресурсов [1];
- с программной точки зрения аппаратная поддержка многопоточности более трудоемка для программного обеспечения [2];

- проблема планирования потоков;
- специфика использования. Вручную настроенные программы на ассемблере, использующие расширения MMX или AltiVec и выполняющие предварительные выборки данных, не страдают от потерь кэша или неиспользуемых вычислительных ресурсов. Таким образом, такие программы не выигрывают от аппаратной многопоточности и действительно могут видеть ухудшенную производительность из-за конкуренции за общие ресурсы.

Однако несмотря на количество недостатков, перечисленных выше, многопоточная парадигма имеет большой потенциал на сегодняшний день и при должном написании кода позволяет значительно ускорить однопоточные алгоритмы.

Цель лабораторной работы

Целью данной лабораторной работы является изучение и реализация параллельных вычислений.

Задачи лабораторной работы

В рамках выполнения работы необходимо решить следующие задачи:

- изучить понятие параллельных вычислений;
- реализовать последовательный и параллельный алгоритм перемножения матриц;
- сравнить временные характеристики реализованных алгоритмов экспериментально.

1 Аналитическая часть

1.1 Описание задачи

Пусть даны две прямоугольные матрицы

$$A_{lm} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{l1} & a_{l2} & \dots & a_{lm} \end{pmatrix}, \quad B_{mn} = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \dots & b_{mn} \end{pmatrix}, \tag{1.1}$$

тогда матрица C

$$C_{ln} = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{l1} & c_{l2} & \dots & c_{ln} \end{pmatrix}, \tag{1.2}$$

где

$$c_{ij} = \sum_{r=1}^{m} a_{ir} b_{rj} \quad (i = \overline{1,l}; j = \overline{1,n}) \tag{1.3}$$

будет называться произведением матриц A и B.

В данной лабораторной работе стоит задача распараллеливания алгоритма Винограда. Так как каждый элемент матрицы C вычисляется независимо от других и матрицы A и B не изменяются, то для параллельного вычисления произведения, достаточно просто равным образом распределить элементы матрицы C между потоками.

1.2 Вывод

Обычный алгоритм перемножения матриц независимо вычисляет элементы матрицы-результата, что дает большое количество возможностей для реализации параллельного варианта алгоритма.

2 Конструкторская часть

На рисунке 2.1 представлена схема обычного алгоритма перемножения матриц (без распараллеливания). На рисунках 2.2 и представлены схема распараллеливания алгоритма умножения матриц.

2.1 Схемы алгоритмов

На рисунке 2.4 представленна схема с параллельным выполнением первого цикла (по строкам матрицы)

2.2 Вывод

На основе теоретических данных, полученных из аналитического раздела, была построена схема стандартного алгоритма умножения матриц, а так же после разделения алгоритма на этапы были предложены 2 схемы параллельного выполнения данных этапов.

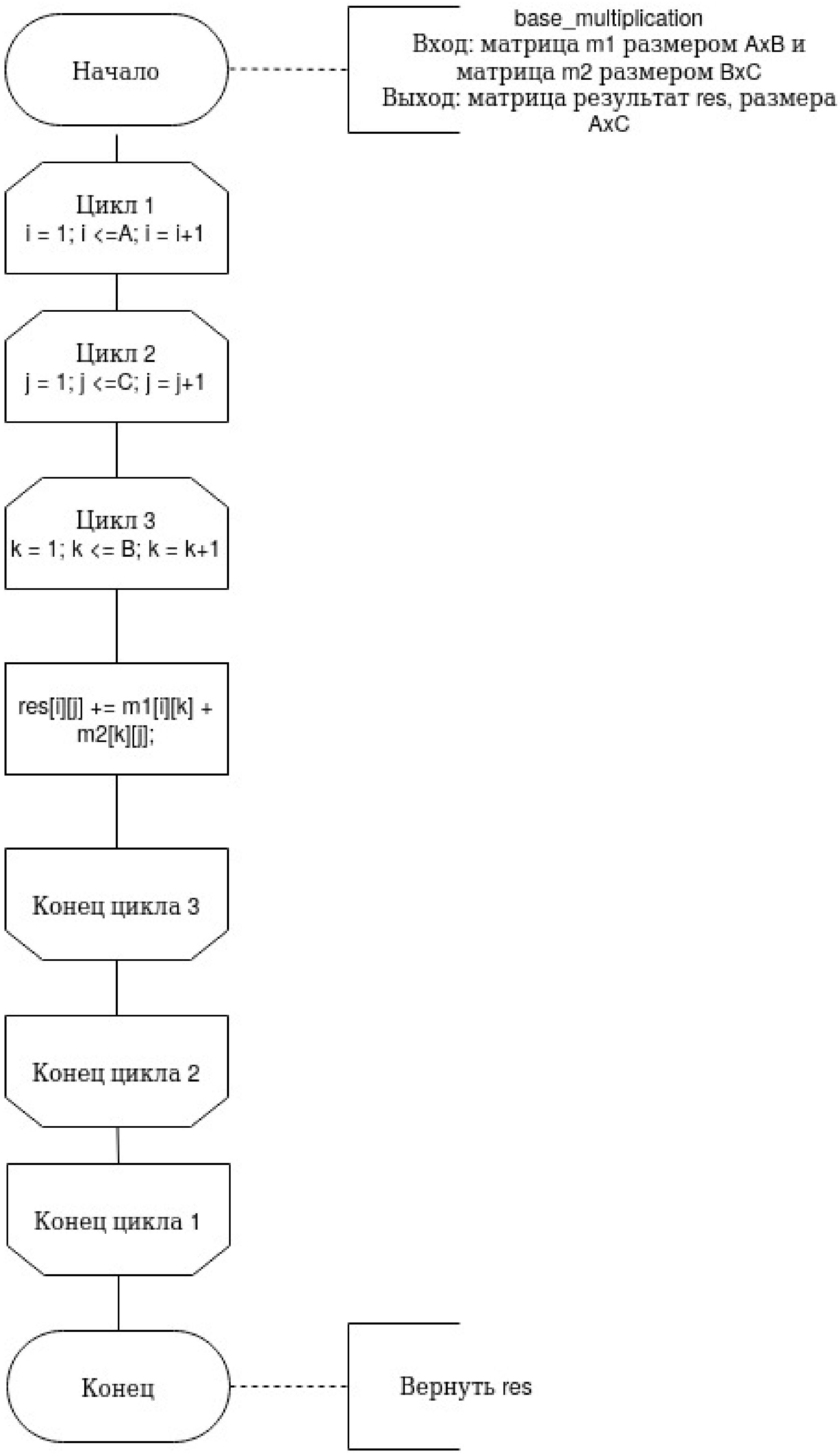

Рис. 2.1: Схема стандартного алгоритма умножения матриц.

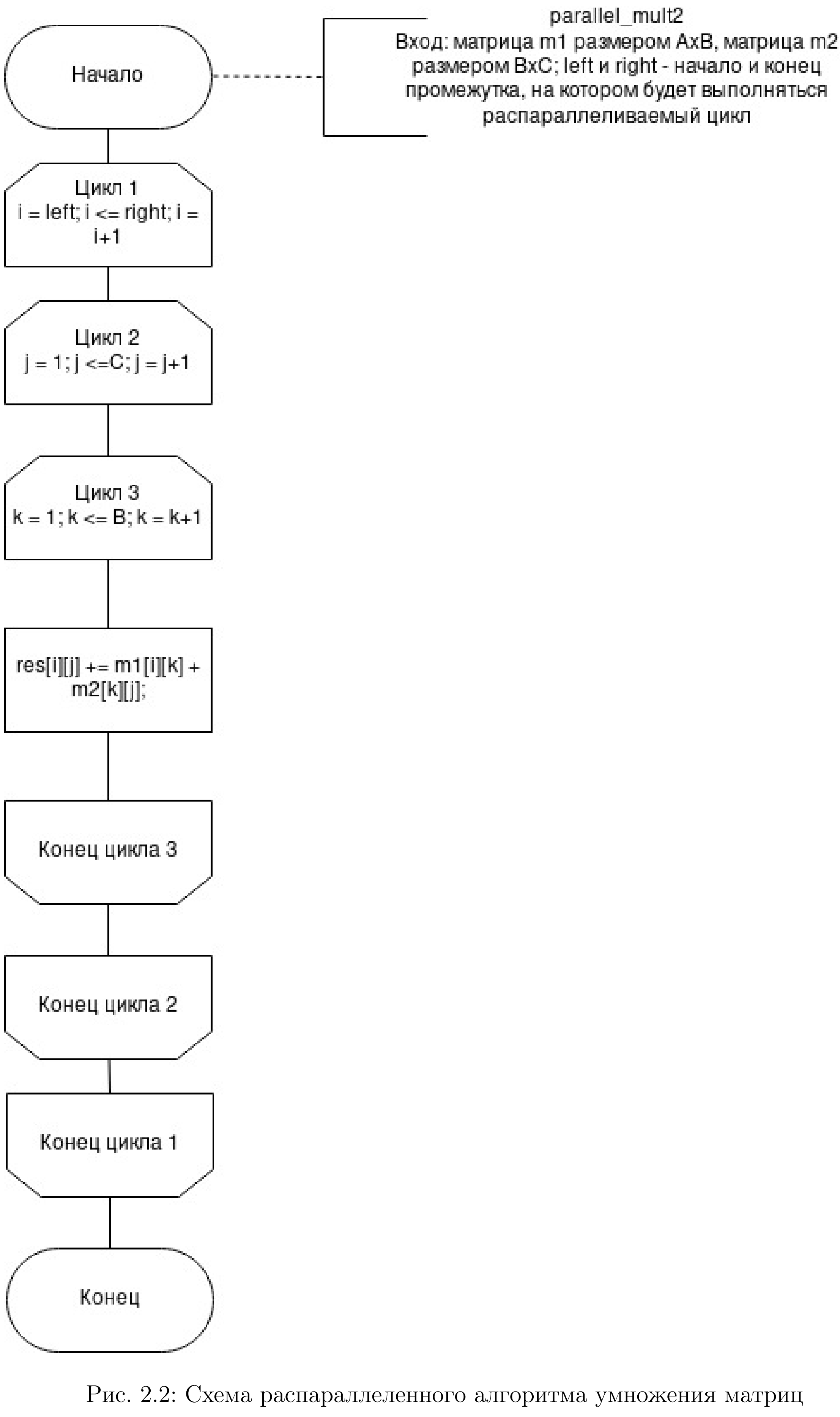

Рис. 2.2: Схема распараллеленного алгоритма умножения матриц

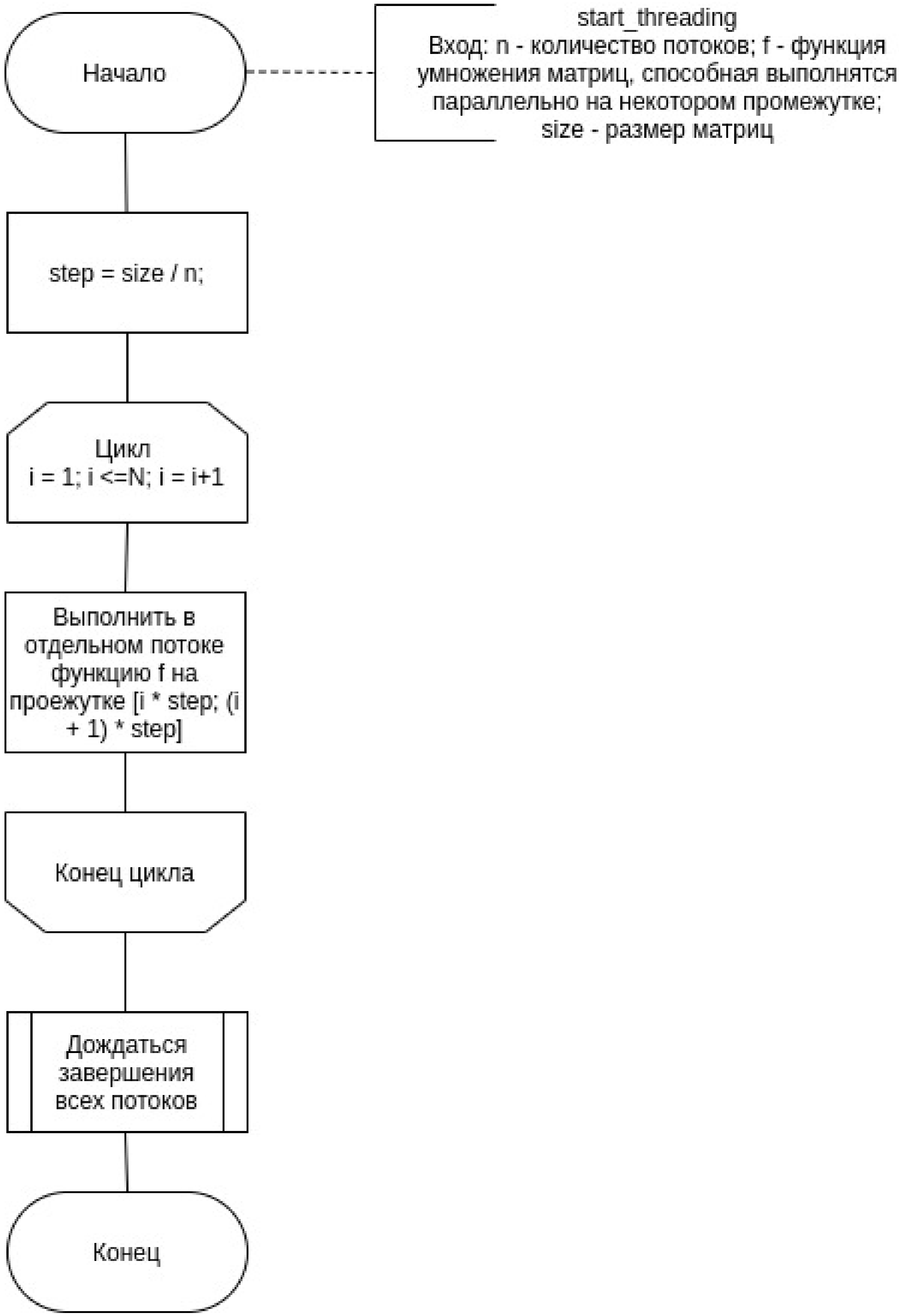

Рис. 2.3: Функция создания потоков и запуска параллельных реализация умножения матриц

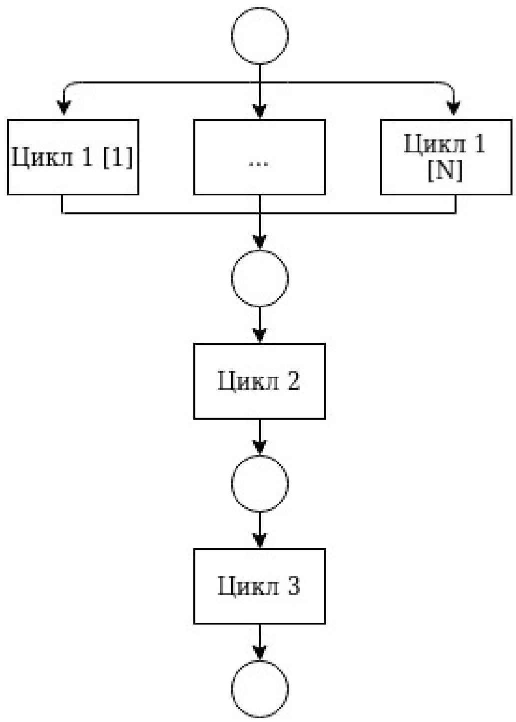

Рис. 2.4: Схема с параллельным выполнением первого цикла

3 Технологическая часть

В данном разделе приведены средства реализации и листинги кода.

3.1 Требование к ПО

К программе предъявляется ряд требований:

- на вход подаются размеры 2 матриц, а также их элементы;
- на выходе — матрица, которая является результатом умножения входных матриц.

3.2 Средства реализации

Для реализации ПО я выбрал язык программирования C++. Данный выбор обусловлен высокой скоростью работы языка, а так же наличия инструментов для создания и эффективной работы с потоками.

3.3 Реализация алгоритмов

В листингах 3.1 - 3.3 приведена реализация расмотренных ранее алгоритмов перемножения матриц. В листинге 3.4 приведена реализация функции создания и распределения потоков.

Листинг 3.1: Функция умножения матриц обычным способом

```
void multiMatrix(double **a, double **b, double **c, int row, int col) {
    for (int i = 0; i < row; i++) {
        for (int j = 0; j < col; j++) {
            c[i][j] = 0;
            for (int k = 0; k < col; k++)
                c[i][j] += a[i][k] * b[k][j];
        }
    }
}
```

Листинг 3.2: Функция умножения матриц параллельно. language

```
void iteration(double **a, double **b, double **c, int row, int col, int
    th_number, int th_amount) {
```

```
    for (int i = th_number; i < row; i += th_amount) {
        for (int j = 0; j < col; j++) {
            c[i][j] = 0;
            for (int k = 0; k < col; k++)
                c[i][j] += a[i][k] * b[k][j];
        }
    }
}
```

Листинг 3.3: Функция создания потоков

```
void multiMatrixParallel(double **a, double **b, double **c, int row, int
    col, int n) {
    std::thread workers[n];
    for (int i = 0; i < n; i++) {
        workers[i] = std::thread(iteration, a, b, c, row, col, i, n);
    }
    for (int i = 0; i < n; i++) {
        workers[i].join();
    }
}
```

3.4 Тестовые данные

В таблице 3.1 приведены тесты для функций, реализующих параллельное и обычное умножение матриц. Все тесты пройдены успешно.

Первая матрица	Вторая матрица	Ожидаемый результат
$\begin{pmatrix}1 & 2 & 3\\1 & 2 & 3\\1 & 2 & 3\end{pmatrix}$	$\begin{pmatrix}1 & 2 & 3\\1 & 2 & 3\\1 & 2 & 3\end{pmatrix}$	$\begin{pmatrix}6 & 12 & 18\\6 & 12 & 18\\6 & 12 & 18\end{pmatrix}$
$\begin{pmatrix}1 & 2 & 2\\1 & 2 & 2\end{pmatrix}$	$\begin{pmatrix}1 & 2\\1 & 2\\1 & 2\end{pmatrix}$	$\begin{pmatrix}5 & 10\\5 & 10\end{pmatrix}$
$\begin{pmatrix}2\end{pmatrix}$	$\begin{pmatrix}2\end{pmatrix}$	$\begin{pmatrix}4\end{pmatrix}$
$\begin{pmatrix}1 & -2 & 3\\1 & 2 & 3\\1 & 2 & 3\end{pmatrix}$	$\begin{pmatrix}-1 & 2 & 3\\1 & 2 & 3\\1 & 2 & 3\end{pmatrix}$	$\begin{pmatrix}0 & 4 & 6\\4 & 12 & 18\\4 & 12 & 18\end{pmatrix}$

Таблица 3.1: Тестирование функций

3.5 Вывод

В данном разделе были разработаны исходные коды алгоритмов: обычный способ умножения матриц и два способа параллельного перемножения матриц.

4 Исследовательская часть

4.1 Технические характеристики

Ниже приведены технические характеристики устройства, на котором было проведено тестирование ПО:

- Операционная система: Debian [3] Linux [4] 11 «bullseye» 64-bit.
- Оперативная память: 12 GB.
- Процессор: Intel(R) Core(TM) i5-3550 CPU @ 3.30GHz [5].

4.2 Время выполнения алгоритмов

В таблице 4.1 приведено сравнение двух реализаций параллельного умножения матриц с разным количеством потоков при перемножении матриц размером 512. В таблице 4.2 приведено сравнение однопоточной реализации и многопоточных (с предварительным транспонированием и без) (на четырёх потоках).

Таблица 4.1: Таблица времени выполнения параллельных алгоритмов, при размерах перемножаемых матриц 512 (в тиках)

Размер матрицы	1 поток	2 потока	4 потока	8 потоков	16 потоков
128	21 752	16 242	12 477	16 472	20 233
256	172 266	127 088	85 094	172 544	186 473
512	1 552 812	1 245 364	943 807	1 713 305	1 720 823
1024	26 584 110	18 450 302	12 547 271	19 919 997	21 114 917

4.3 Вывод

Наилучшее время параллельные алгоритмы показали на 4 потоках, что соответствует количеству логических ядер компьютера, на котором проводилось тестирование. На матрицах размером 512 на 512, параллельные алгоритмы улучшают время обычной (однопоточной) реализации перемножения матриц примерно в 1.7 раза. При количестве потоков, большее чем

4, параллельная реализация замедляет выполнение (в сравнении с 4 потоками). Кроме этого, если рассматривать транспонирование матриц, то можно получить улучшение в среднем на 15% быстрее обычного параллельного умножения. Это можно объяснить кэшированием элементов.

Заключение

В рамках данной лабораторной работы была достигнута её цель: изучены паралелльные вычисления. Также выполнены следующие задачи:

- было изучено понятие параллельных вычислений;
- были реализованы обычный и параллельный алгоритм перемножения матриц;
- было произведено сравнение временных характеристик реализованных алгоритмов экспериментально.

Параллельные реализации алгоритмов выигрывают по скорости у обычной (однопоточной) реализации перемножения двух матриц. Наиболее эффективны данные алгоритмы при количестве потоков, совпадающем с количеством логических ядер компьютера. Так же стоит отметить, что при дополнительном транспонировании второй матрицы, можно уменьшить время выполнения параллельных алгоритмов в среднем на 15%, что объясняется кэшированием элементов.

Литература

[1] Mario Nemirovsky D. M. T. Multithreading Architecture // Morgan and Claypool Publishers. 2013.

[2] Olukotun K. Chip Multiprocessor Architecture — Techniques to Improve Throughput and Latency // Morgan and Claypool Publishers. 2007. p. 154.

[3] Debian – универсальная операционная система [Электронный ресурс]. Режим доступа: https://www.debian.org/. Дата обращения: 20.09.2020.

[4] Linux – Getting Started [Электронный ресурс]. Режим доступа: https://linux.org. Дата обращения: 20.09.2020.

[5] Процессор Intel® Core™ i5-3550 [Электронный ресурс]. Режим доступа: https://ark.intel.com/content/www/ru/ru/ark/products/65516/intel-core-i5-3550-processor-6m-cache-up-to-3-70-ghz.html. Дата обращения: 20.09.2020.