



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»

Отчёт по лабораторной работе №4 по дисциплине "Анализ алгоритмов"

Тема Параллельное умножение матриц

Студент Романов А.В.

Группа ИУ7-53Б

Оценка (баллы) _____

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

Москва — 2020 г.

Оглавление

Введение	2
1 Аналитическая часть	4
1.1 Описание задачи	4
1.2 Вывод	4
2 Конструкторская часть	5
2.1 Схемы алгоритмов	5
2.2 Вывод	5
3 Технологическая часть	11
3.1 Требование к ПО	11
3.2 Средства реализации	11
3.3 Реализация алгоритмов	11
3.4 Тестовые данные	13
3.5 Вывод	13
4 Исследовательская часть	15
4.1 Технические характеристики	15
4.2 Время выполнения алгоритмов	15
4.3 Вывод	15
Заключение	17
Литература	17

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.

Задачи лабораторной работы

В рамках выполнения работы необходимо решить следующие задачи:

- изучить понятие параллельных вычислений;
- реализовать последовательный и 2 параллельных реализации алгоритма перемножения матриц;
- сравнить временные характеристики реализованных алгоритмов экспериментально.

1 | Аналитическая часть

1.1 Описание задачи

Пусть даны две прямоугольные матрицы

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

тогда матрица C

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

где

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

будет называться произведением матриц A и B .

В данной лабораторной работе стоит задача распараллеливания алгоритма Винограда по 2 схемам. 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 2.3 представлены схемы первого и второго варианта распараллеливания алгоритма умножения матриц. На рисунке 2.4 показана схема функции, запускающей вышеописанные функции на нужных промежутках.

2.1 Схемы алгоритмов

На рисунке 2.5 представлена схема с параллельным выполнением первого цикла (по строкам матрицы), а на рисунке 2.6 – с параллельным выполнением второго цикла (по столбам 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.

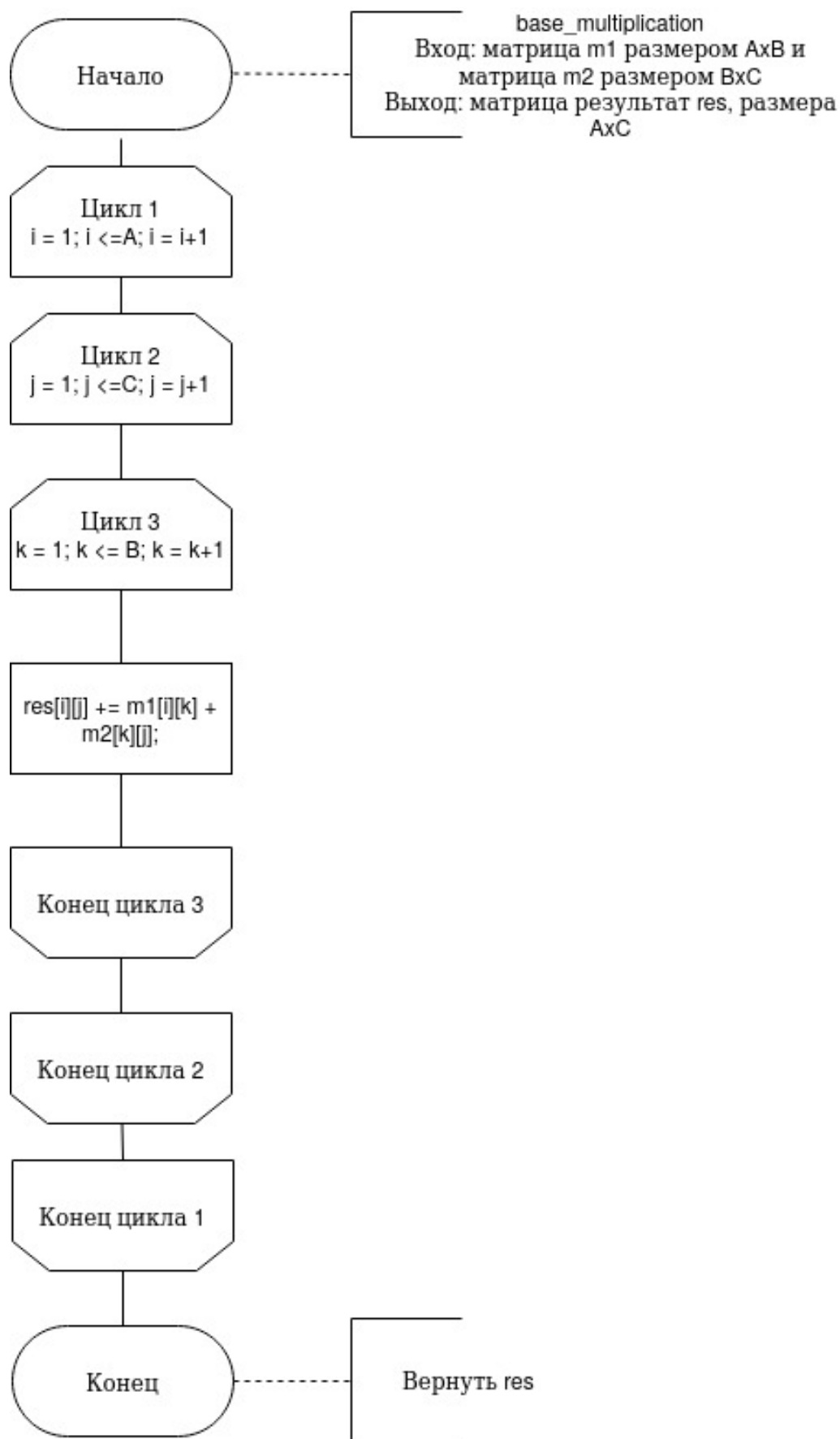


Рис. 2.1: Схема стандартного алгоритма умножения матриц.

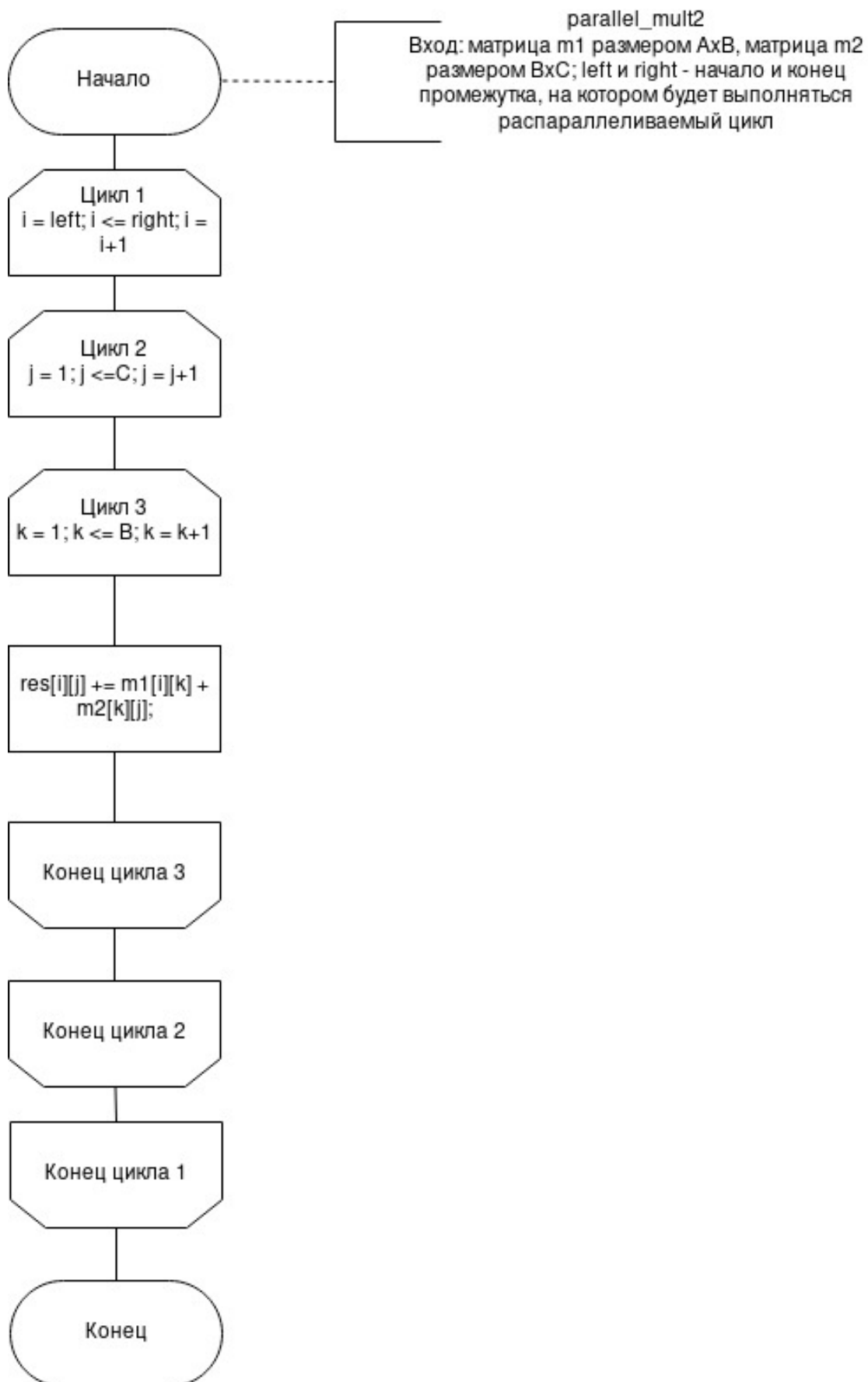


Рис. 2.2: Схема распараллеленного алгоритма умножения матриц, способ №1.

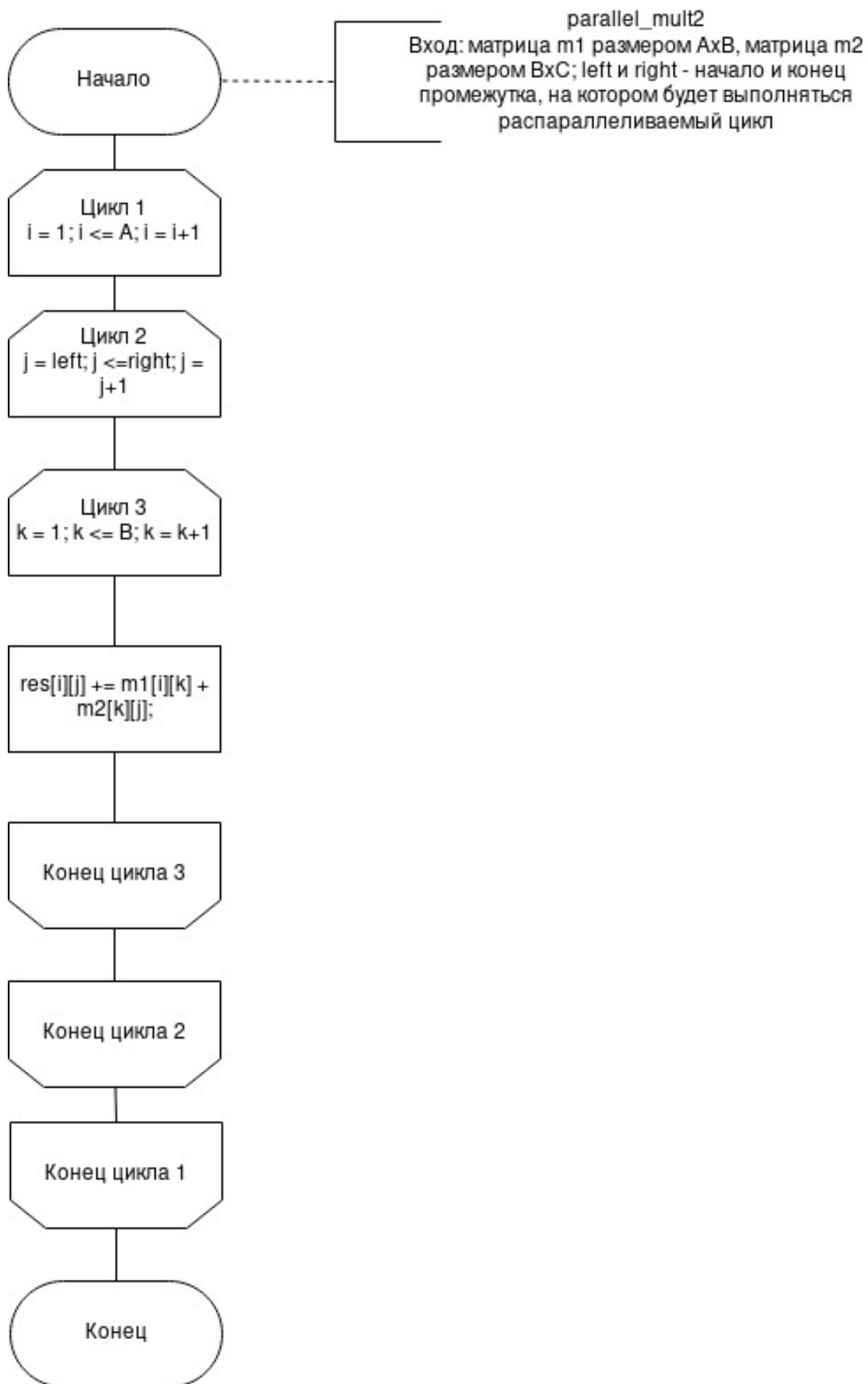


Рис. 2.3: Схема распараллеленного алгоритма умножения матриц, способ №2.

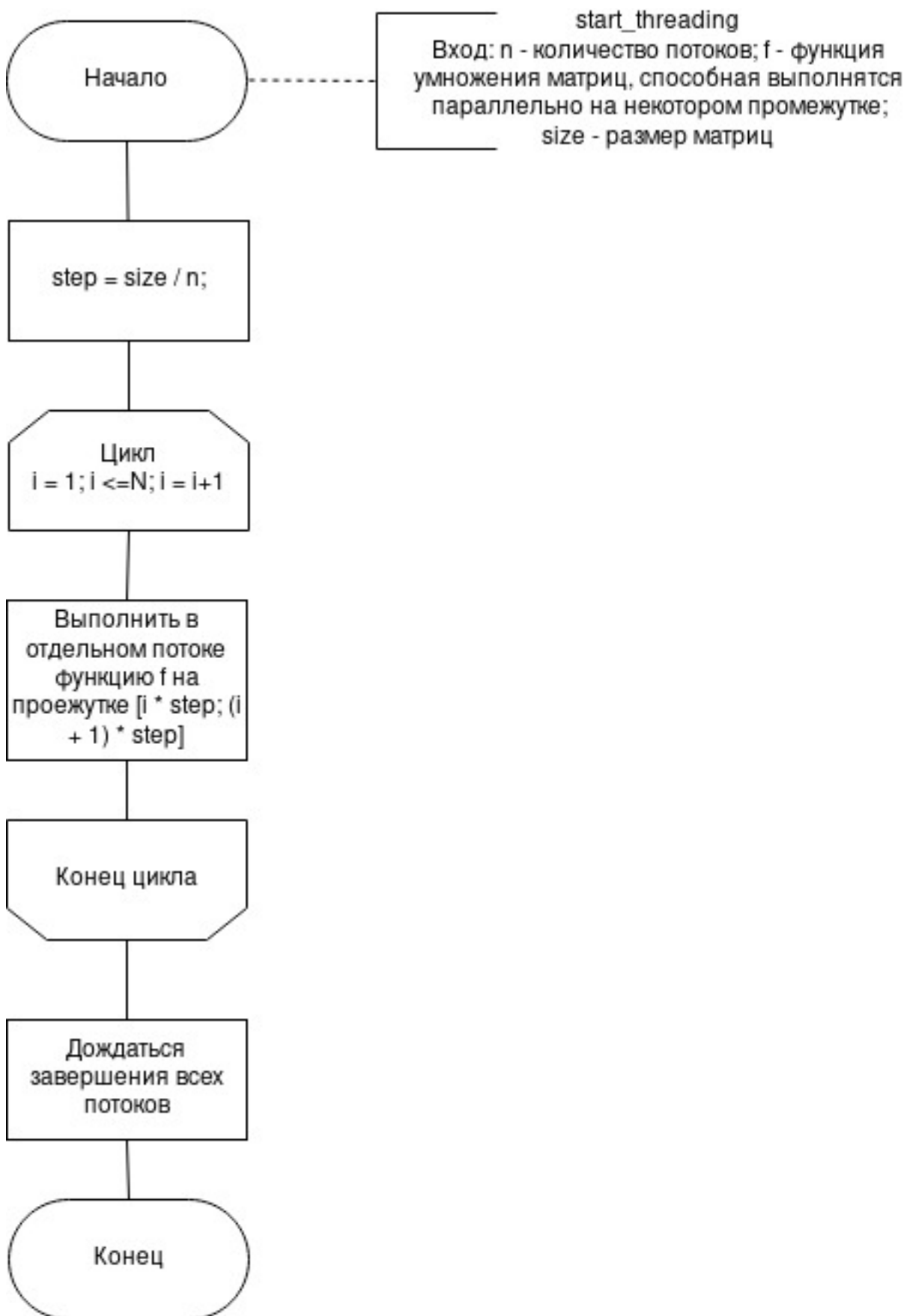


Рис. 2.4: Функция создания потоков и запуска параллельных реализация умножения матриц

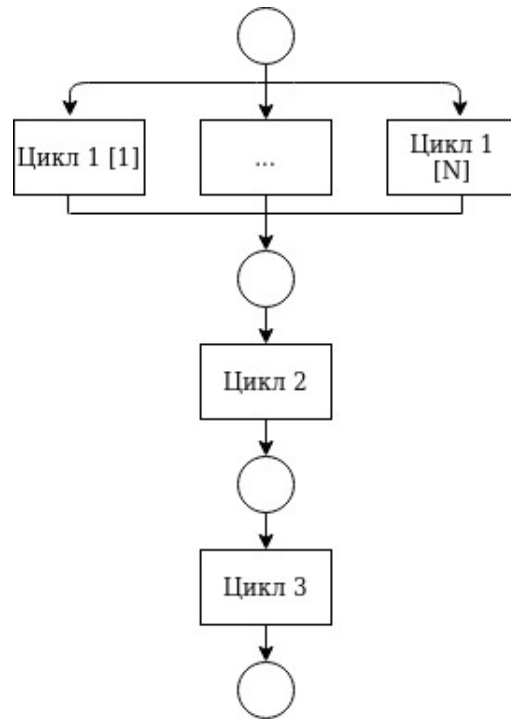


Рис. 2.5: Схема с параллельным выполнением первого цикла

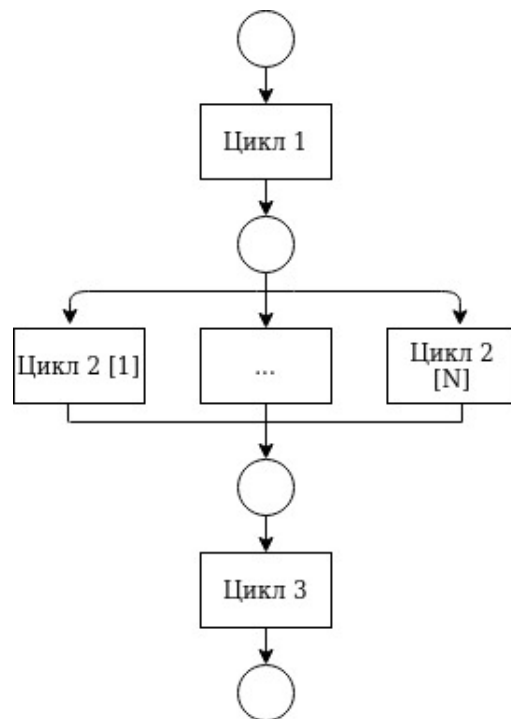


Рис. 2.6: Схема с параллельным выполнением второго цикла

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 Си [3]. 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 рассмотренных ранее алгоритмов перемножения матриц. В листинге 3.4 приведена реализация функции создания и распределения потоков.

Листинг 3.1: Функция умножения матриц обычным способом

```
1 void base_multiplication(args_t *args) {  
2     for (int i = 0; i < N; i++) {  
3         for (int j = 0; j < K; j++) {  
4             args->res[i][j] = 0;  
5             for (int k = 0; k < M; k++) {  
6                 args->res[i][j] += args->m1[i][k] * args->m2[k][j];  
7             }  
8         }  
9     }  
10 }
```

Листинг 3.2: Функция умножения матриц параллельно. Способ №1

```
1 void *parallel_multiplication1(void *args) {
```

```

2 pthread_args_t *argsp = (args_t *)args;
3
4 int row_start = argsp->tid * (argsp->size / argsp->cnt_threads);
5 int row_end = (argsp->tid + 1) * (argsp->size / argsp->cnt_threads);
6
7 for (int i = row_start; i < row_end; i++) {
8     for (int j = 0; j < K; j++) {
9         argsp->mult_args->res[i][j] = 0;
10        for (int k = 0; k < M; k++) {
11            argsp->mult_args->res[i][j] += argsp->mult_args->m1[i][k] * argsp->
                mult_args->m2[k][j];
12        }
13    }
14 }
15
16 return NULL;
17 }

```

Листинг 3.3: Функция умножения матриц параллельно. Способ №2

```

1 void *parallel_multiplication2(void *args) {
2     pthread_args_t *argsp = (args_t *)args;
3
4     int col_start = argsp->tid * (argsp->size / argsp->cnt_threads);
5     int col_end = (argsp->tid + 1) * (argsp->size / argsp->cnt_threads);
6
7     for (int i = 0; i < N; i++) {
8         for (int j = col_start; j < col_end; j++) {
9             argsp->mult_args->res[i][j] = 0;
10            for (int k = 0; k < M; k++) {
11                argsp->mult_args->res[i][j] += argsp->mult_args->m1[i][k] * argsp->
                    mult_args->m2[k][j];
12            }
13        }
14    }
15
16    return NULL;
17 }

```

Листинг 3.4: Функция создания потоков

```

1 int start_threading(args_t *args, const int cnt_threads, const int type) {
2
3     pthread_t *threads = malloc(cnt_threads * sizeof(pthread_t));
4
5     if (!threads) {
6         return ALLOCATE_ERROR;
7     }
8
9
10    pthread_args_t *args_array = malloc(sizeof(pthread_args_t) * cnt_threads);

```

```

11
12  if (!args_array) {
13      free(threads);
14      return ALLOCATE_ERROR;
15  }
16
17  for (int i = 0; i < cnt_threads; i++) {
18      args_array[i].mult_args = args;
19      args_array[i].tid = i;
20      args_array[i].size = N;
21      args_array[i].cnt_threads = cnt_threads;
22
23      if (type == 1) {
24          pthread_create(&threads[i], NULL, parallel_multiplication1, &
25              args_array[i]);
26      } else {
27          pthread_create(&threads[i], NULL, parallel_multiplication2, &
28              args_array[i]);
29      }
30  }
31
32  for (int i = 0; i < cnt_threads; i++) {
33      pthread_join(threads[i], NULL);
34  }
35
36  free(args_array);
37  free(threads);
38
39  return OK;
40 }

```

3.4 Тестовые данные

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

3.5 Вывод

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

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

Таблица 3.1: Тестирование функций

4 | Исследовательская часть

4.1 Технические характеристики

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

- Операционная система: Debian [4] Linux [5] 11 «bullseye» 64-bit.
- Оперативная память: 12 GB.
- Процессор: Intel(R) Core(TM) i5-3550 CPU @ 3.30GHz [6].

4.2 Время выполнения алгоритмов

В таблице 4.1 представлено сравнение простого алгоритма (однопоточного) и параллельных алгоритмах, которые исполняются на 4 потоках. В таблице 4.2 приведено сравнение двух реализаций параллельного умножения матриц с разным количеством потоков при перемножении матриц размером 512.

Таблица 4.1: Таблица времени выполнения простого и параллельных алгоритмов (на 4 потоках) перемножения матриц (в тиках)

Размер матрицы	Обычный	Параллельный 1	Параллельный 2
64	11 569 478	4 423 500	4 399 005
128	59 224 135	16 626 572	17 706 987
256	354 152 337	98 670 198	96 640 191
512	2 719 059 760	755 595 299	751 307 775
1024	26 717 252 997	7 347 757 065	7 489 450 509

4.3 Вывод

Наилучшее время параллельные алгоритмы показали на 4 потока, что соответствует количеству логических ядер компьютера, на котором проводилось тестирование. На матрица размеров 512 на 512, параллельные алгоритмы улучшают время обычной (однопоточной) реализации перемножения матриц примерно в 3.5 раза. При количестве потоков, большее чем 4, параллельная реализация замедляет выполнение (в сравнении с 4 потоками).

Таблица 4.2: Таблица времени выполнения параллельных алгоритмов, при размерах перемножаемых матриц 512 (в тиках)

Количество потоков	Параллельный 1	Параллельный 2
1	2 808 685 166	2 811 309 156
2	1 418 843 814	1 420 505 354
4	754 354 852	749 388 813
8	761 861 636	753 869 394
16	756 535 197	755 651 844
32	777 543 386	766 654 800

Заключение

В рамках данной лабораторной работы:

- было изучено понятие параллельных вычислений;
- были реализованы обычный и 2 параллельных реализаций алгоритма перемножения матриц;
- было произведено сравнение временных характеристик реализованных алгоритмов экспериментально.

Параллельные алгоритмы значительно выигрывают по времени аналогичные простые (однопоточные) реализации. Наиболее эффективны данные алгоритмы при количестве потоков, совпадающем с количеством логических ядер компьютера. Так, например, на матрицах размером 512 на 512, удалось улучшить время выполнения алгоритма умножения матриц в 3.6 раза (в сравнении с однопоточной реализацией).

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] The C99 Standard. Режим доступа: <http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1256.pdf>. Дата обращения: 16.09.2020.
- [4] Debian — универсальная операционная система [Электронный ресурс]. Режим доступа: <https://www.debian.org/>. Дата обращения: 20.09.2020.
- [5] Linux — Getting Started [Электронный ресурс]. Режим доступа: <https://linux.org>. Дата обращения: 20.09.2020.
- [6] 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.