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

Название: Параллельные вычисления

Дисциплина: Анализ алгоритмов

Студент ИУ7-52Б _____ А.С. Пронин
(Группа) (Подпись, дата) (И.О. Фамилия)

Преподователь _____ Л.Л. Волкова
(Подпись, дата) (И.О. Фамилия)

Москва, 2021

Содержание

Введение 3
1 Аналитический раздел 5
1.1 Описание задачи 5
1.2 Вывод 5
2 Констукторский раздел 6
2.1 Схемы алгоритмов 6
2.2 Вывод 6
3 Технологический раздел 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 Экспериментальный раздел 15
4.1 Технические характеристики 15
4.2 Время выполнения алгоритмов 15
4.3 Вывод 15
Заключение 17
Список литературы 18

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;

— с программной точки зрения аппаратная поддержка многопоточности более трудоемка для программного обеспечения;

— 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 параллельную реализацию подсчета среднего геометрического столюцов матрицы;

— сравнить временные характеристики реализованных алгоритмов экспериментально.

1 Аналитический раздел

1.1 Описание задачи

Пусть дана прямоугольная матрица

$$A_{nm} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{l1} & a_{l2} & \dots & a_{lm} \end{pmatrix}, \tag{1.1}$$

тогда матрица B

$$B_{1n} = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1m} \end{pmatrix}, \tag{1.2}$$

где

$$B_{1j} = \sqrt[1/n]{\prod_{i=1}^{n} a_{ij}} \quad (j = \overline{1,m}) \tag{1.3}$$

будет называться средним геометрическим столбцов матрицы A.

В данной лабораторной работе стоит задача распараллеливания алгоритма получения среднего геометрического столбцов матрицы. Так как каждый столбец матрицы B вычисляется независимо от других и матрица A не изменяется, то для параллельного вычисления среднего геометрического, достаточно просто равным образом распределить столбцы матрицы B между потоками.

1.2 Вывод

Обычный алгоритм получения среднего геометрического от ряда чисел в столбце матрицы независимо вычисляет элементы матрицы-результата, что дает большое количество возможностей для реализации параллельного варианта алгоритма.

2 Конструкторский раздел

На рисунке 2.1 представлена схема обычного алгоритма получения среднего геометрического столбцов матрицы (без распараллеливания). На рисунке 2.2 представлена схема варианта распараллеливания алгоритма получения среднего геометрического столбцов матрицы. На риснуке 2.3 показана схема функции, определяющая границы циклов для каждого из потоков.

2.1 Схемы алгоритмов

На рисунке 2.4 представленна схема с параллельным выполнением первого цикла

2.2 Вывод

На основе теоретических данных, полученных из аналитического раздела, была построена схема алгоритма получения среднего геометрического столбцов матрицы, а так же после разделения алгоритма на этапы была предложена схема параллельного выполнения данных этапов.

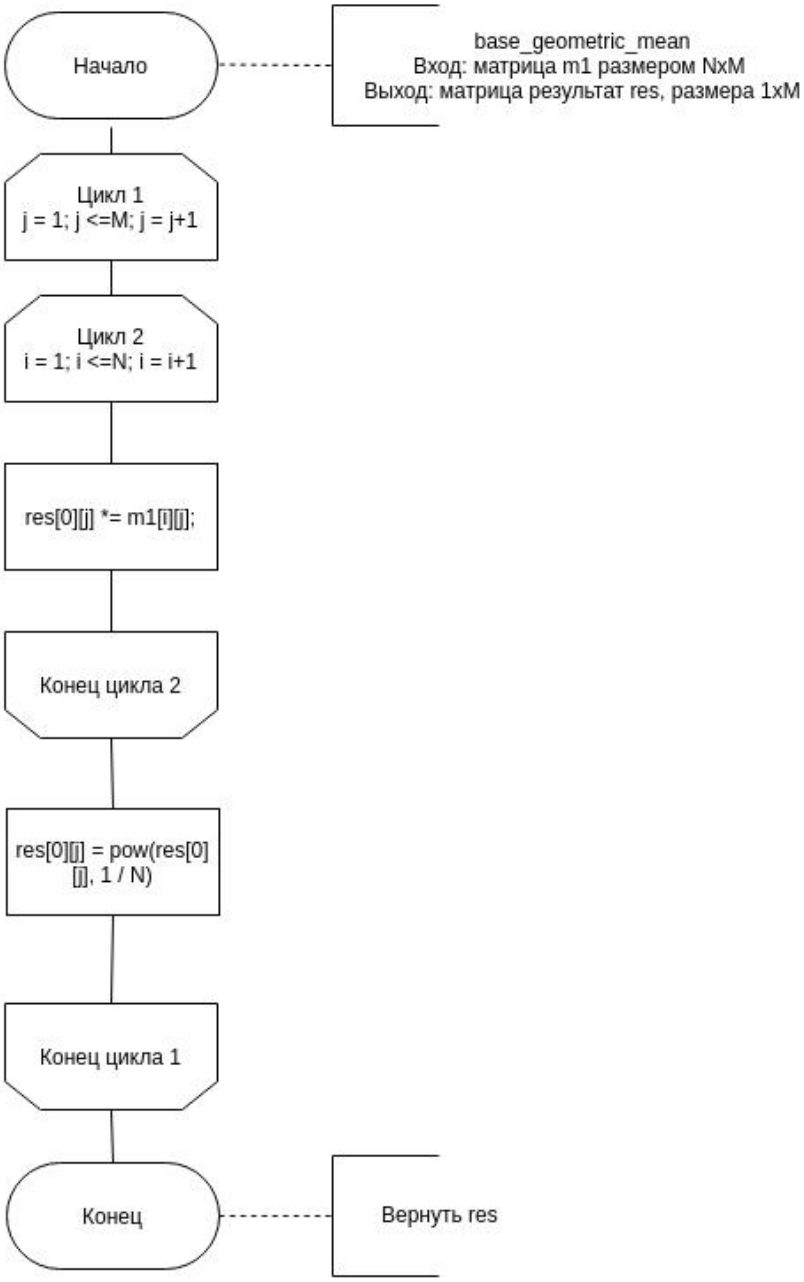

Рисунок 2.1 — Схема стандартного алгоритма получения среднего геометрического столбцов матрицы.

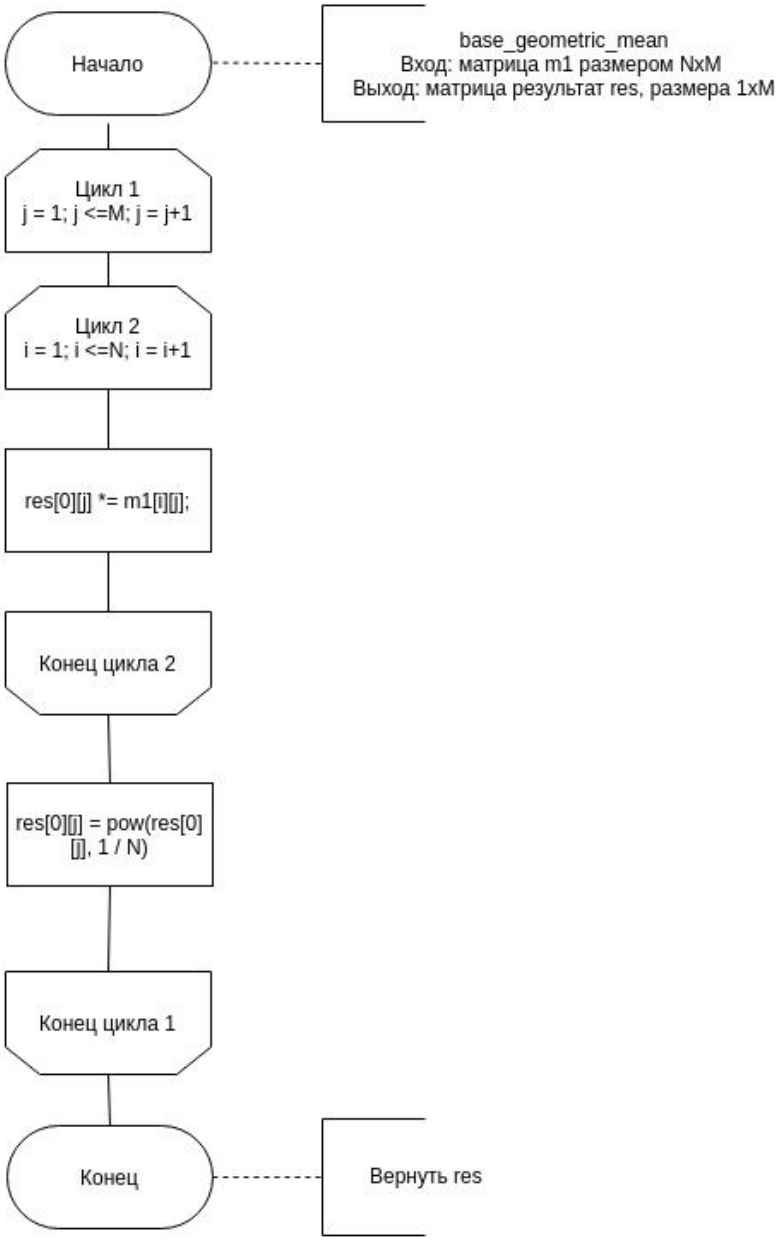

Рисунок 2.2 — Схема распараллеленного алгоритма получения среднего геометрического столбцов матрицы.

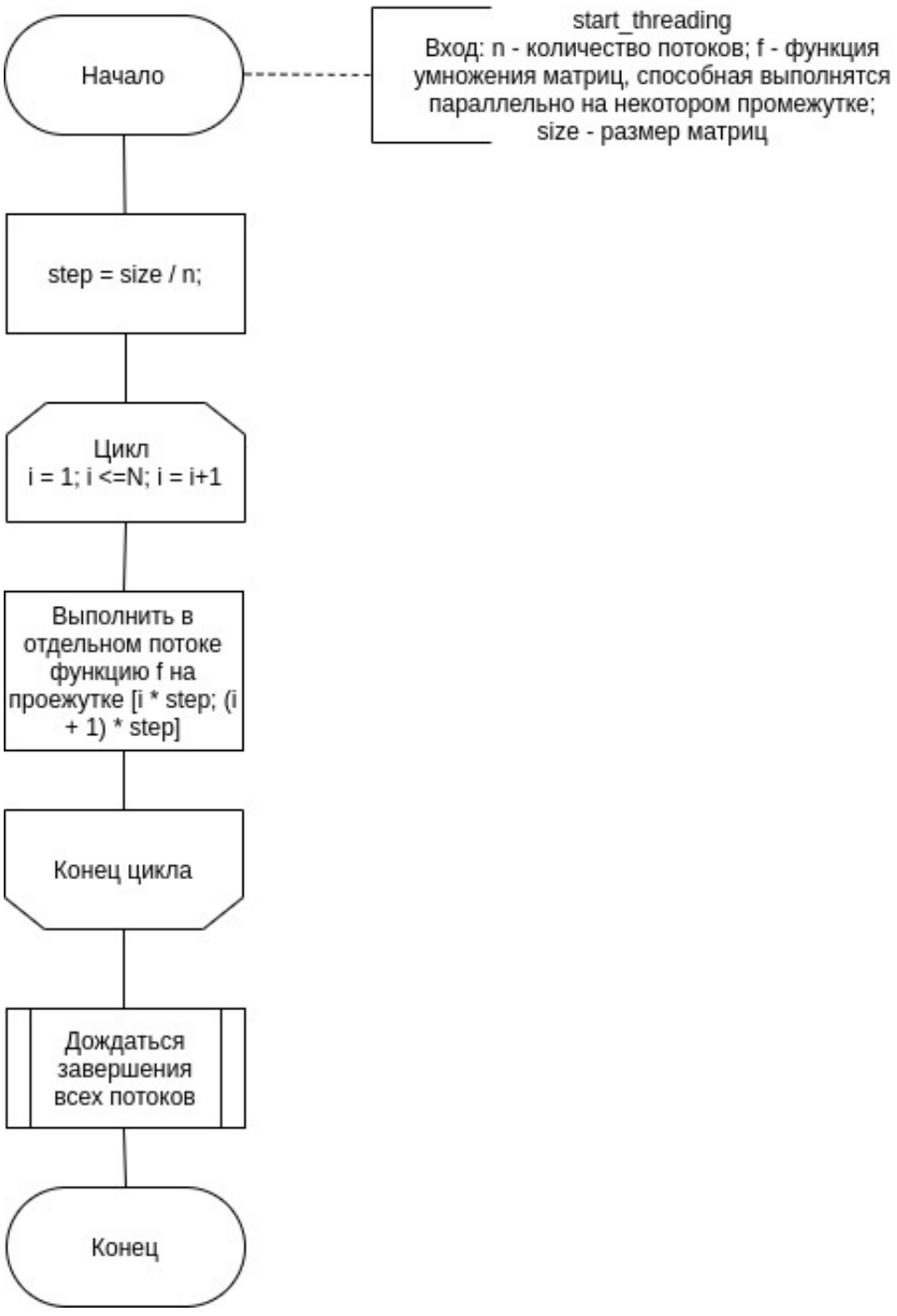

Рисунок 2.3 — Функция создания потоков и запуска параллельных реализаций получения среднего геометрического столбцов матрицы

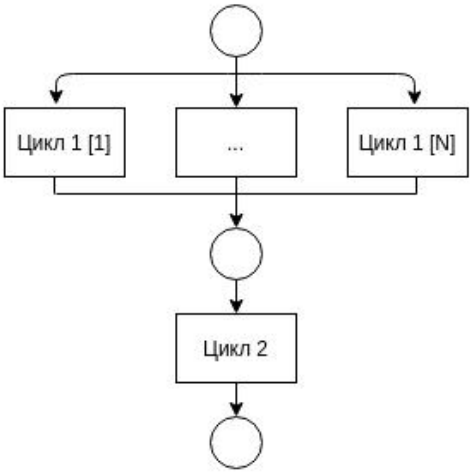

Рисунок 2.4 — Схема с параллельным выполнением цикла

3 Технологический раздел

В данном разделе приведены средства реализации и листинги кода.

3.1 Требование к ПО

К программе предъявляется ряд требований:

— на вход подаются размеры матрицы, а также её элементы;

— на выходе — матрица, которая является результатом получения среднего геометрического столбцов входной матрицы.

3.2 Средства реализации

Для реализации ПО я выбрал язык программирования Си [1]. 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 - 3.2 приведена реализация расмотренных ранее алгоритмов получения среднего геометрического столбцов матрицы. В листинге 3.3 приведена реализация функции создания и распределения потоков.

Листинг 3.1 — Функция получения среднего геометрического столбцов матрицы обычным способом

```
1  void base_geometric_mean(args_t *args) {
2      for (int j = 0; j < M; j++) {
3          int ind_tmp_val = 1;
4          for (int i = 0; i < N; i++) {
5              args->res[ind_tmp_val][j] *= args->m1[i][j];
6              if (i % 10 == 0 && i != 0) {
7                  double n = N;
8                  args->res[ind_tmp_val][j] = pow(args->res[ind_tmp_val][j],
                       static_cast<double>(1) / n);
9                  ind_tmp_val++;
10             }
11         }
12         for (int ind = 1; ind < N / 10 + 2; ind++)
13             args->res[0][j] *= args->res[ind][j];
14     }
15 }
```

Листинг 3.2 — Функция получения среднего геометрического столбцов матрицы параллельно.

```
1   void *parallel_geometric_mean_by_columns(void *args) {
2       pthread_args_t *all_data = (pthread_args_t *)args;
3
4       int col_start = all_data->thread_id * (all_data->matrix_size /
            all_data->cnt_threads);
5       int col_end = (all_data->thread_id + 1) * (all_data->matrix_size /
            all_data->cnt_threads);
6
7       for (int j = col_start; j < col_end; j++) {
8           int ind_tmp_val = 1;
9           for (int i = 0; i < N; i++) {
10              all_data->matrix_container->res[ind_tmp_val][j] *=
                    all_data->matrix_container->m1[i][j];
11              if (i % 10 == 0 && i != 0) {
12                  double n = N;
13                  all_data->matrix_container->res[ind_tmp_val][j] =
14                          pow(all_data->matrix_container->res[ind_tmp_val][j],
                                static_cast<double>(1) / n);
15                  ind_tmp_val++;
16              }
17          }
18          for (int ind = 1; ind < N / 10 + 2; ind++)
19              all_data->matrix_container->res[0][j] *=
                    all_data->matrix_container->res[ind][j];
20      }
21      return NULL;
22  }
```

Листинг 3.3 — Функция создания потоков

```
1   int start_threading(args_t *args, const int cnt_threads, const int type) {
2       pthread_t *threads = (pthread_t *)malloc(cnt_threads * sizeof(pthread_t));
3
4       if (!threads) {
5           fprintf(stderr, "Ошибка при выделении памяти. Файл: %s\nСтрока: %d\n",
                __FILE__, __LINE__);
6           return ALLOCATE_ERROR;
7       }
8
9       pthread_args_t *args_array = (pthread_args_t *)malloc(sizeof(pthread_args_t)
            * cnt_threads);
10
11      if (!args_array) {
12          free(threads);
```

```
13              fprintf(stderr, "Ошибка при выделении памяти: %d\nФайл: %s\n", __LINE__,
                    __FILE__);
14              return ALLOCATE_ERROR;
15          }
16
17          for (int i = 0; i < cnt_threads; i++) {
18              args_array[i].matrix_container = args;
19              args_array[i].thread_id = i;
20              args_array[i].matrix_size = N;
21              args_array[i].cnt_threads = cnt_threads;
22
23              switch (type) {
24                  case 1:
25                      pthread_create(&threads[i], NULL,
                            parallel_geometric_mean_by_columns, &args_array[i]);
26                      break;
27              }
28          }
29
30          for (int i = 0; i < cnt_threads; i++) {
31              pthread_join(threads[i], NULL);
32          }
33
34          free(args_array);
35          free(threads);
36
37          return OK;
38      }
```

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.

Матрица	Ожидаемый результат
$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 5 & 6 & 5 \\ 7 & 8 & 9 & 6 \end{pmatrix}$	$\begin{pmatrix} 3.036589 & 4.308869 & 5.451362 & 4.932424 \end{pmatrix}$
$\begin{pmatrix} 3 & 4 & 2 \\ 27 & 16 & 8 \end{pmatrix}$	$\begin{pmatrix} 9 & 8 & 4 \end{pmatrix}$
$\begin{pmatrix} 4 \end{pmatrix}$	$\begin{pmatrix} 4 \end{pmatrix}$
$\begin{pmatrix} 0 & 0 & 0 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 0 & 0 \end{pmatrix}$

Таблица 3.1 — Тестирование функций

4 Исследовательский раздел

4.1 Технические характеристики

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

— Операционная система: Windows 10
— Оперативная память: 36 GB.
— Процессор: Intel Core i7-7700K

4.2 Время выполнения алгоритмов

В таблице 4.1 приведено сравнение реализации параллельного получения среднего геометрического столбцов матрицы с разным количеством потоков при размере исходной матрицы 512. В таблице 4.2 приведено сравнение однопоточной реализации и многопоточных (на четырёх потоках).

Таблица 4.1 — Таблица времени выполнения параллельных алгоритмов, при размере матрицы 512 (в тиках)

Количество потоков	Паралльная реализация алгоритма
1	16 352 377
2	8 413 109
4	6 617 493
8	5 469 821
16	5 144 998
24	5 825 791
32	7 378 193

4.3 Вывод

Наилучшее время параллельные алгоритмы показали на 16 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 пример-

Таблица 4.2 — Таблица времени выполнения простого и параллельных алгоритмов (на 4 потоках) перемножения матриц (в тиках)

Размер матрицы	Обычный	Параллельный
64	209 360	766 202
128	775 438	1 057 966
256	2 575 187	2 485 468
512	10 710 964	4 797 693
1024	60 078 099	17 591 850

но в 2.5 раза. При количестве потоков, большее чем 16, параллельная реализация замедляет выполнение (в сравнении с 16 потоками).

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 параллельная реализации алгоритма получения среднего геометрического столбцов матрицы;

— было произведено сравнение временных характеристик реализованных алгоритмов экспериментально.

Параллельные реализации алгоритмов выигрывают по скорости у обычной (однопоточной) реализации получения среднего геометрического столбцов матрицы. 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 2.5 раза (в сравнении с однопоточной реализацией).

Список литературы

1. The C99 Standard. Режим доступа:http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1256.pdf. Дата обращения: 3.10.2021.

2. Debian – универсальная операционная система [Электронный ресурс]. Режим доступа: https://www.debian.org/. Дата обращения: 3.10.2021.

3. Linux – Getting Started [Электронный ресурс]. Режим доступа: https://linux.org. Дата обращения: 3.10.2021.

4. 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: 3.10.2021.