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»
ДИСЦИПЛИНА «Анализ алгоритмов»

Лабораторная работа № 4

Дисциплина Анализ алгоритмов
Тема Разработка параллельных алгоритмов
Студент Боренко А. Д.
Группа ИУ7-52Б
Оценка (баллы) _____
Преподаватель Волкова Л.Л.

Москва — 2022г.

Содержание

Введение

Параллельные вычисления — способ организации компьютерных вычислений, при котором программы разрабатываются как набор взаимодействующих вычислительных процессов, работающих параллельно (одновременно).

Причины для использования параллелизма:

1. разделение обязанностей;
2. повышение производительности.

Разделение обязанностей почти всегда приветствуется при разработке программ: если сгруппировать взаимосвязанные и разделить несвязанные части кода, то программа станет проще для понимания и тестирования и, стало быть, будет содержать меньше ошибок.

Повысить производительность можно 2 способами: разбить задачу на части и запустить их параллельно, уменьшив тем самым общее время выполнения и воспользоваться имеющимся параллелизмом для решения более крупных задач, например, обрабатывать не один файл за раз, а сразу два, десять или двадцать, - способы называются распараллеливание по задачам и по данным, соответственно.

Целью данной работы является разработка и исследования параллельных алгоритмов нахождения среднего арифметического матрицы. Задачами данной лабораторной являются:

1. реализация последовательного алгоритма нахождения среднего арифметического матрицы;
2. реализация распараллеленного алгоритма нахождения среднего арифметического матрицы;

3. сравнительный анализ реализованных алгоритмов;

4. описание и обоснование полученных результатов в отчете о выполненной лабораторной работе, выполненного как расчетно-пояснительная записка к работе.

1. Аналитическая часть

В данном разделе будут рассмотрен алгоритм нахождения среднего арифметического матрицы и идея его параллельной реализации.

1.1 Последовательный алгоритм нахождения среднего арифметического матрицы

Последовательно суммируются все элементы матрицы, а после делятся на их количество - строки * столбцы.

1.2 Распараллеленный алгоритм нахождения среднего арифметического матрицы

Для увеличения производительности приведенный алгоритм можно распараллелить. Рассмотрим два способа распараллеливания: по строкам и столбца. В первом случае каждый поток обрабатывает элементы своих строк (эти строки обрабатываются только им). Во втором каждый поток обрабатывает элементы своих столбцов.

На рисунке 1.1 показан принцип распараллеливания по строкам.

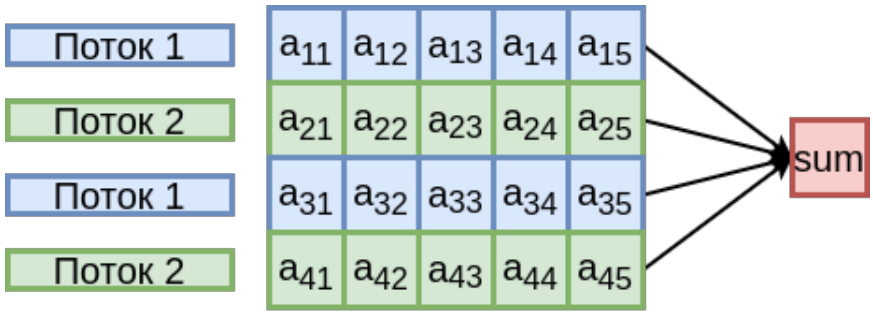

Рис. 1.1: Принцип работы распараллеленого алгоритма

1.3 Вывод аналитической части

В данной работе стоит задача реализации следующих алгоритмов: последовательного алгоритма нахождения среднего арифметического матрицы, распараллеленого по строкам алгоритма нахождения среднего арифметического матрицы, распараллеленого по столбцам алгоритма нахождения среднего арифметического матрицы. Необходимо сравнить алгоритмы умножения матриц по эффективности по времени.

2. Конструкторская часть

В данном разделе представлены схемы алгоритмов. Так же будут опи- саны пользовательские структуры данных, приведены классы эквивалентности для тестирования реализуемого ПО.

2.1 Схемы алгоритмов

2.1.1 Схема последовательного алгоритма нахождения среднего арифметического матрицы

На рисунке 2.1 показана схема последовательного алгоритма умножения матриц.

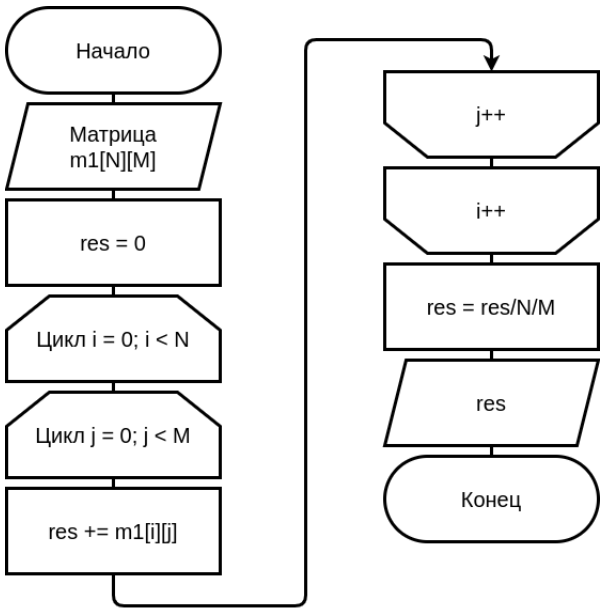

Рис. 2.1: Схема последовательного алгоритма

2.1.2 Схема распараллеленного по строкам алгоритма нахождения среднего арифметического матрицы

На рисунке 2.2 показана схема распараллеленного по строкам алгоритма умножения матриц.

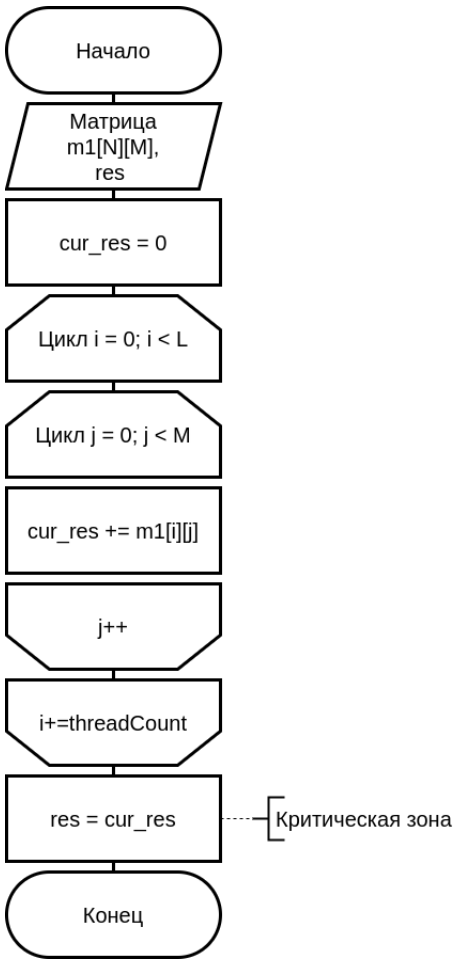

Рис. 2.2: Схема распараллеленого по строкам алгоритма

2.1.3 Схема распараллеленного по столбцам алгоритма нахождения среднего арифметического матрицы

На рисунке 2.3 показана схема распараллеленного по столбцам алгоритма умножения матриц.

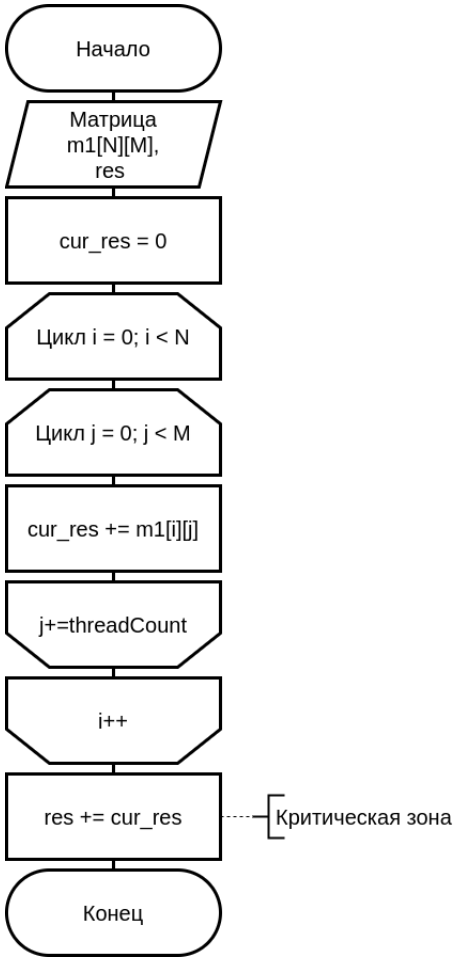

Рис. 2.3: Схема распараллеленного по столбцам алгоритма

2.1.4 Схема алгоритма управления потоками

На рисунке 2.4 показана схема алгоритма управления потоками.

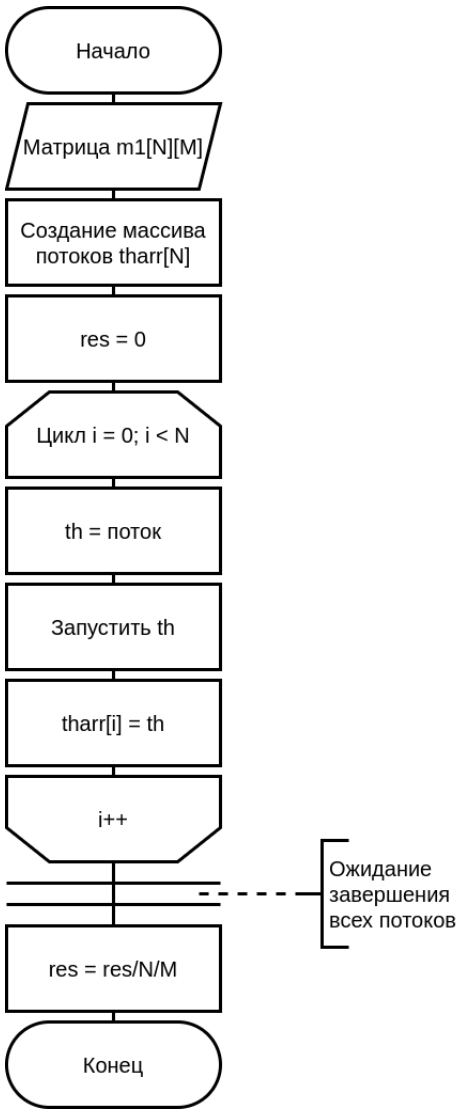

Рис. 2.4: Схема алгоритма управления потоками

2.2 Структуры данных

При реализации приведенных алгоритмов потребуется тип данных: матрица.

2.3 Тестирование

Для алгоритма умножения матриц можно выделить следующие классы эквивалентности:

1. матрицы $n \cdot m$ (где $n, m \in N$);

2. матрицы, где $n = 0$ или $m = 0$.

2.4 Вывод конструкторской части

На основе данных, полученных в аналитическом разделе, были построены схемы используемых алгоритмов, выделены необходимые для реализации структуры данных и методы тестирования.

3. Технологическая часть

3.1 Требования к ПО

Требования к программному обеспечению:

1. наличие меню для реализации выбора пользователя;

2. на вход подается матрица;

3. результат: число -среднее арифметическое матрицы.

3.2 Выбор языка программирования

Был выбран язык C++, поскольку в нем используются нативные потоки. Средой разработки выбрана Visual Studio Code.

3.3 Структуры данных

На листинге 3.1 представлено описание структуры матрицы.

Листинг 3.1: Структура матрицы

```
1 struct matrix_t{
2     double ** elems;
3     int rows;
4     int columns;
5 };
```

3.4 Реализация алгоритмов

На листинге 3.2 представлена реализация последовательного алгоритма нахождения среднего арифметического матрицы.

Листинг 3.2: Реализация последовательного алгоритма нахождения среднего арифметического матрицы

```
void standart_average(matrix_t &m1, double &res)
{
    res = 0;
    for (int i = 0; i < m1.rows; i++){
        for (int j = 0; j < m1.columns; j++){
            res += m1.elems[i][j];
        }
    }
    res = res / m1.rows/m1.columns;
}
```

На листинге 3.3 представлена реализация распараллеленого по строкам алгоритма нахождения среднего арифметического матрицы.

Листинг 3.3: Реализация распараллеленого по строкам алгоритма нахождения среднего арифметического матрицы

```
void parall_row_average(double &res, double **elems, int rows, int columns,
    int index, int step)
{
    double r = 0;
    for (int i = index; i < rows; i+=step){
        for (int j = 0; j < columns; j++){
            r += elems[i][j];
        }
    }
    MuTeX.lock();
    res += r;
    MuTeX.unlock();
}
```

На листинге 3.4 представлена реализация распараллеленого по столбцам алгоритма нахождения среднего арифметического матрицы.

Листинг 3.4: Реализация распараллеленого по столбцам алгоритма нахождения среднего арифметического матрицы

```
void parall_column_average(double &res, double **elems, int rows, int
    columns, int index, int step)
{
    double r = 0;
    for (int i = 0; i < rows; i++){
        for (int j = index; j < columns; j+=step){
            r += elems[i][j];
        }
    }
    MuTeX.lock();
    res += r;
    MuTeX.unlock();
}
```

3.5 Тестирование

Модульные тесты

Таблица 3.1 – Тесты

№	Ввод	Вывод
1	0 0	Нет элементов.
3	1 0	Нет элементов.
2	2 2 2 2 2 2	2

Тесты пройдены.

3.6 Вывод технологической части

Были реализованы исследуемые алгоритмы, программа прошла тесты и удовлетворяет требованиям.

4. Экспериментальная часть

Оценка качества работы алгоритмов. Экспериментальное сравнение работы различных алгоритмов нахождения среднего арифметического матрицы (зависимость времени выполнения от размерности матриц).

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование:

1. процессор: Intel® Core™ i3-7100U CPU @ 2.40GHz × 4;
2. память: 11,6 GiB;
3. операционная система: Ubuntu 20.04.1 LTS.

4.2 Примеры работы

На рисунках 4.1, 4.2, 4.3 показаны примеры работы.

```
МЕНЮ
        1. Ручное тестирование
        2. Автотестирование
        3. График
        4. Выход
Ваш выбор: 1
Введите размер матрицы[rowsxcolumns]:
строки: 2
столбцы: 2
2 2
2 2
Ответ:
Стандартный алгоритм: 2
Распараллеленый по строкам (4 потока): 2
Распараллеленый по столбцам (4 потока): 2
Матрица: 2x2
2 2
2 2
```

Рис. 4.1: Пример 1

```
МЕНЮ
        1. Ручное тестирование
        2. Автотестирование
        3. График
        4. Выход
Ваш выбор: 1
Введите размер матрицы[rowsxcolumns]:
строки: 0
столбцы: 0
В матрице нет элементов. Нельзя расчитать среднее арифметическое.
```

Рис. 4.2: Пример 2

```
Таблица (в сек) (Выбор 2)
  Длина|     stand|1     стр.|1    стол.|2     стр.|2    стол.|4     стр.|4    стол.|8     стр.|8    стол.|16    стр.|16   стол.|32    стр.|32   стол.
    100| 0,000061| 0,000117| 0,000114| 0,000089| 0,000091| 0,000110| 0,000106| 0,000240| 0,000220| 0,000443| 0,000464| 0,000938| 0,001014
    200| 0,000168| 0,000408| 0,000400| 0,000220| 0,000226| 0,000193| 0,000246| 0,000308| 0,000314| 0,000502| 0,000719| 0,000956| 0,000984
    300| 0,000360| 0,000913| 0,000875| 0,000443| 0,000434| 0,000541| 0,000525| 0,000578| 0,000465| 0,000596| 0,000593| 0,001250| 0,001268
    400| 0,000643| 0,001469| 0,001458| 0,000752| 0,000769| 0,000521| 0,000606| 0,000808| 0,000840| 0,000858| 0,000842| 0,001087| 0,001160
    500| 0,001170| 0,002230| 0,002269| 0,001150| 0,001221| 0,000704| 0,000824| 0,001209| 0,001325| 0,000975| 0,001197| 0,001330| 0,001601
    600| 0,001674| 0,003223| 0,003208| 0,001635| 0,001752| 0,001031| 0,000970| 0,001683| 0,001679| 0,002703| 0,001784| 0,001604| 0,002437
    700| 0,002030| 0,004326| 0,004305| 0,002178| 0,002272| 0,001286| 0,001593| 0,002267| 0,002287| 0,001812| 0,002496| 0,001971| 0,003593
    800| 0,002610| 0,005652| 0,005687| 0,002922| 0,002910| 0,001750| 0,001998| 0,002553| 0,002668| 0,002340| 0,003618| 0,002550| 0,004759
    900| 0,003303| 0,007122| 0,007118| 0,003630| 0,004041| 0,002390| 0,002110| 0,003224| 0,003124| 0,002982| 0,004482| 0,003174| 0,005949
   1000| 0,004170| 0,008773| 0,008743| 0,004610| 0,004488| 0,003147| 0,002866| 0,003382| 0,003976| 0,003351| 0,005450| 0,003705| 0,007063
```

Рис. 4.3: Пример 3

4.3 Замеры времени

Таблица 4.1 содержит результаты замеров времени при 1, 2, 4, 8, 16, 32 потоках

Таблица 4.1 – Замеры времени (в 10^{-2}сек)

Длина	stand	1 стр.	1 стол.	2 стр.	2 стол.	4 стр.	4 стол.	8 стр.	8 стол.	16 стр.	16 стол.	32 стр.	32 стол.
100	0.0079	0.0067	0.0065	0.0064	0.0064	0.0087	0.0088	0.0196	0.0198	0.0412	0.0443	0.0839	0.0855
200	0.0168	0.0199	0.0194	0.0126	0.0126	0.0138	0.0133	0.0239	0.0249	0.0447	0.0467	0.0898	0.0914
300	0.0358	0.0414	0.0411	0.0232	0.0228	0.0207	0.0252	0.0332	0.0383	0.0495	0.0507	0.0946	0.1039
400	0.0639	0.0693	0.0719	0.0364	0.0376	0.0319	0.0351	0.0558	0.0645	0.0577	0.0771	0.1022	0.1075
500	0.0997	0.1057	0.1052	0.0543	0.0544	0.0459	0.0518	0.0760	0.0920	0.0738	0.1063	0.0999	0.1463
600	0.1441	0.1504	0.1496	0.0777	0.0792	0.0626	0.0685	0.1003	0.1317	0.0893	0.1630	0.1138	0.2116
700	0.1963	0.2028	0.2017	0.1045	0.1073	0.0840	0.0927	0.1293	0.1749	0.1198	0.2155	0.1382	0.2990
800	0.2572	0.2636	0.2618	0.1334	0.1403	0.1135	0.1197	0.1735	0.2156	0.1383	0.2923	0.1621	0.4061
900	0.3254	0.3320	0.3301	0.1698	0.1783	0.1297	0.1509	0.1992	0.2516	0.1661	0.3670	0.1910	0.5182
1000	0.4030	0.4085	0.4085	0.2068	0.2152	0.1608	0.1856	0.2247	0.3022	0.2117	0.4611	0.2243	0.6299

На рисунках 4.5 и 4.4 показаны графические результаты сравнения исследуемых алгоритмов по времени.

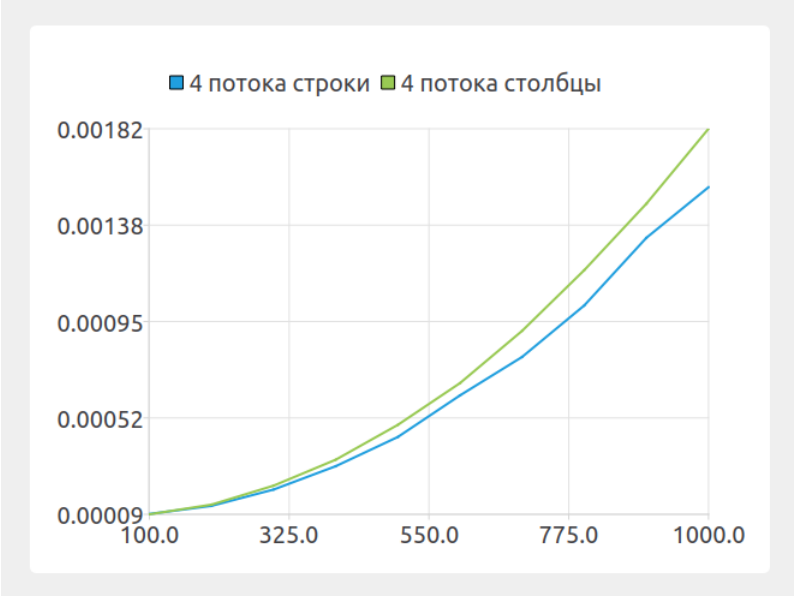

Рис. 4.4: Сравнение распараллеленого по строкам и столбцам алгоритмов

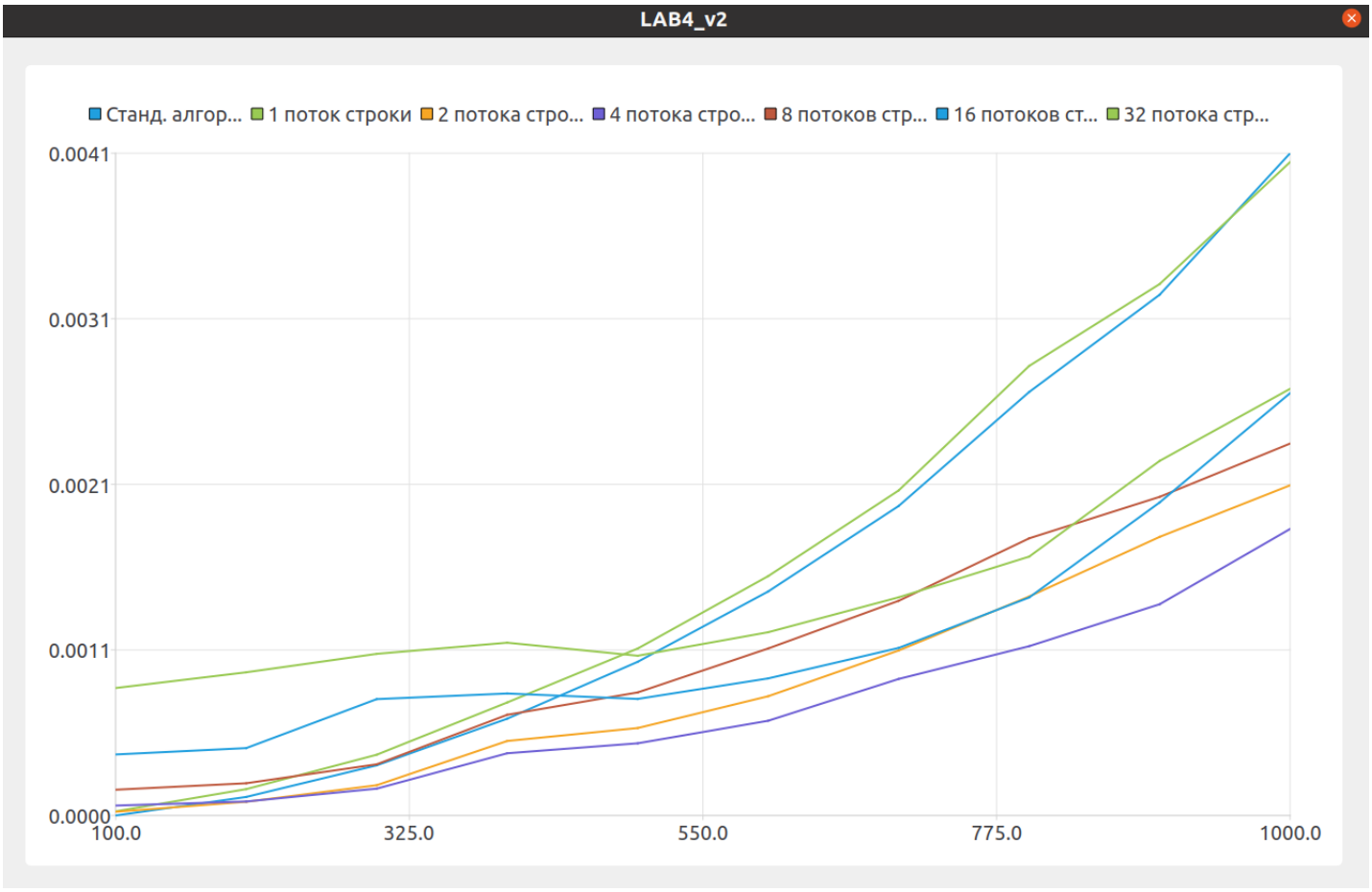

Рис. 4.5: Сравнение стандартного и распараллеленых по строкам алгоритмов

4.4 Сравнительный анализ алгоритмов

По результатам экспериментов можно заключить следующее:

1. при относительно небольшом размере матриц (менее 100x100) использование потоков для уменьшения времени исполнения нецелесообразно, так как накладные расходы времени на управление потоками и mutexами больше, чем выигрыш от параллельного выполнения выполнения вычислений;

2. использование по крайней мере двух потоков даёт ощутимый выигрыш по времени по сравнению с однопоточной версией алгоритма;

3. использование одного потока в многопоточных версиях алгоритма проигрывает по времени по сравнению с однопоточной версией алгоритма, что объясняется накладными расходами времени на управление потоками и mutex-ами;

4. использование 8 и 32 потоков показывает результат по времени несколько хуже, чем при 4 потоках, из чего следует, что увеличение потоков даёт выигрыш по времени лишь до достижения определённого количества, так как появляются большие накладные затраты по времени для управления большим количеством потоков и mutex-ов;

5. параллельные версии алгоритма выполняются за приблизительно одинаковое время при одном потоке. Однако, использование большего количества потоков выявляет, что многопоточность по строкам быстрее многопоточности по столбцам;

6. наиболее быстродейственно алгоритм действует на 4 потоках, что равно количеству логических процессоров на испытуемом компьютере/

4.5 Вывод экспериментальной части

Таким образом, в задачах, где выполняются несколько действий параллельно и независят от результатов предыдущих действий, выгодно использовать потоки.

Оптимальным количеством потоков является число равное кол-ву процессоров на испытуемом компьютере.

Заключение

В данной работе были изучены алгоритмы нахождения среднего арифметического матрицы. Получены практические навыки реализации параллельных алгоритмов. Проведён сравнительный анализ алгоритмов по времени. Экспериментально подтверждены различия в эффективности алгоритмов с указанием лучших и худших случаев. Цель работы достигнута, решены поставленные задачи.

Список литературы

[1] Gratzer George A. More Math Into LaTeX. 4th изд. Boston: Birkhauser, 2007.

[2] Debian – универсальная операционная система [Электронный ресурс]. Режим доступа: https://www.debian.org/. Дата обращения: 20.12.2021.

[3] Linux – Getting Started [Электронный ресурс]. Режим доступа: https://linux.org. Дата обращения: 20.12.2021.

[4] Макконнел. Дж. Анализ алгоритмов. Активный обучающий подход. – М. Режим доступа:https://www.linux.org.ru/. 2017. 267 с.

[5] Документация языка C++ 98 [Электронный ресурс]. Режим доступа:http://www.open-std.org/JTC1/SC22/WG21/. Дата обращения: 22.12.2021.