Министерство образования и науки 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»

Отчет по лабораторной работе № 4

По курсу "Анализ Алгоритмов"

Параллельное умножение матриц

Студент:

Турсунов Жасурбек Рустамович

Группа: ИУ7-56Б

Преподователи:

Волкова Лилия Леонидовна
Строганов Юрий Владимирович

Москва, 2020 г.

Содержание

Введение

Целью данной лабораторной работы является изучение возможности параллельных вычислений и использование такого подхода на практике. В данной лабораторной работе рассматривается классический алгоритм умножения матриц и параллельный алгоритм классического алгоритма умножения матриц. Необходимо сравнить зависимость времени работы алгоритма от числа параллельных потоков и размера матриц. Также следует провести сравнение стандартного и параллельного алгоритмов.

1 Аналитическая часть

Матрицей A размера [m * n] называется прямоугольная таблица чисел, функций или алгебраических выражений, содержащая m строк и n столбцов. Числа m и n определяют размер матрицы. Если число столбцов в первой матрицы совпадает с числом строк во второй матрице, то эти матрицы можно перемножить. У произведения будет столько же строк, сколько в первой матрице и столько же столбцов, сколько во второй.

Пусть даны две прямоугольные матрицы A и B размеров [m * n] и [n * k] соответственно. В результате произведения матриц A и B получим матрицу C размера [m * k].[1]

$c_{i,j} = \sum_{r=1}^{n} a_{i,r} \cdot b_{r,j}$ называется произведением матриц A и B.

1.1 Параллельный алгоритм классического умножения матриц

Чтобы улучшить алгоритм, следует распараллелить ту часть алгоритма, которая содержит 3 вложенных цикла. Вычисление результата для каждой строки не зависит от результата выполнения умножения для других строк. Поэтому можно распараллелить часть кода, где происходят эти действия. Каждый поток будет выполнять вычисления определенных строк результирующей матрицы.

1.2 Параллельное программирование

При использовании многопроцессорных вычислительных систем с общей памятью обычно предполагается, что имеющиеся в составе системы процессоры обладают равной производительностью, являются равноправными при доступе к общей памяти, и время доступа к памяти является одинаковым (при одновременном доступе нескольких процессоров к одному и тому же элементу памяти очередность и синхронизация доступа обеспечивается на аппаратном уровне). Многопроцессорные системы подобного типа обычно именуются симметричными мультипроцессорами (symmetric multiprocessors, SMP).

Перечисленному выше набору предположений удовлетворяют также активно развиваемые в последнее время многоядерные процессоры, в которых каждое ядро представляет практически независимо функциони рующее вычислительное устройство. Для общности излагаемого учебного материала для упоминания одновременно и мультипроцессоров и много ядерных процессоров для обозначения одного вычислительного устройства.

Обычный подход при организации вычислений для многопроцессорных вычислительных систем с общей памятью – создание новых параллельных методов на основе обычных последовательных программ, в которых или автоматически компилятором, или непосредственно программистом выделяются участки независимых друг от друга вычислений. Возможности автоматического анализа программ для порождения параллельных вычислений достаточно ограничены, и второй подход является преобладающим. При этом для разработки параллельных программ могут применяться как новые алгоритмические языки, ориентированные на параллельное программирование, так и уже имеющиеся языки, расширенные некоторым набором операторов для параллельных вычислений.

Широко используемый подход состоит и в применении тех или иных библиотек, обеспечивающих определенный программный интерфейс (application programming interface, API) для разработки параллельных программ. В рамках такого подхода наиболее известны Windows Thread API. Однако первый способ применим только для ОС семейства Microsoft Windows, а второй вариант API является достаточно трудоемким для использования и имеет низкоуровневый характер. [2]

1.3 Организация взаимодействия параллельных потоков

Потоки исполняются в общем адресном пространстве параллельной программы. Как результат, взаимодействия параллельных потоков можно организовать через использование общих данных, являющихся доступными для всех потоков. Наиболее простая ситуация состоит в использовании общих данных только для чтения.

В случае же, когда общие данные могут изменяться несколькими потоками, необходимы специальные усилия для организации правильного взаимодействия.

1.4 Вывод

Были рассмотрены алгоритмы классического умножения матриц и возможность его оптимизации с помощью распараллеливания потоков. Была рассмотрена технология параллельного программирования и организация взаимодействия параллельных потоков.

2 Конструкторская часть

Требования к вводу: На вход подаются две матрицы.

Требования к программе:

1. корректное умножение двух матриц;
2. при матрицах разных размеров программа не должнв аварийно завершаться.

2.1 Разработка алгоритмов

В данном разделе будут рассмотрена схема алгоритма:

1. классический алгоритм умножения матриц;

На рисунке 1 представлена схема классического алгоритма умножения матриц.

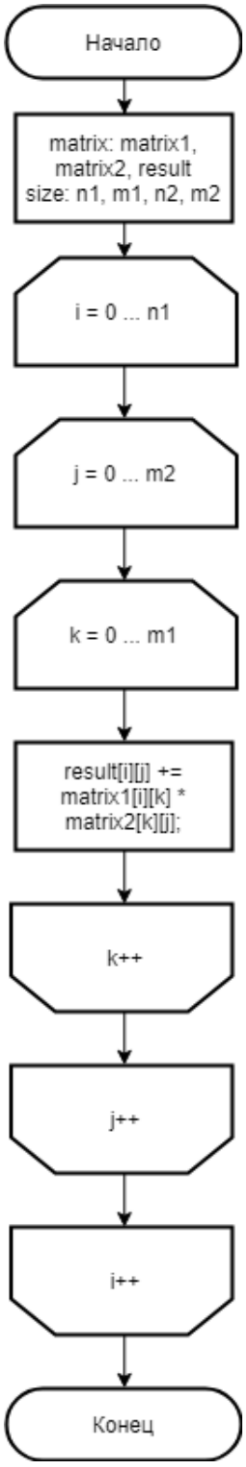

Рис. 1: Схема классического алгоритма умножения матриц

2.2 Распараллеливание программы

Распараллеливание программы должно ускорять время работы. Это достигается за счет реализации в узких участках(например в циклах с большим количеством независимых вычислений). В предложенном алгоритме данным участком будет являться тройной цикл поиска результата.

2.3 Вывод

В данном разделе были рассмотрена схема алгоритма классического умножения матриц и способ ее распараллеливания

3 Технологическая часть

В данном разделе будут рассмотрены требования к программному обеспечению, средства реализации и представлен листинг кода.

3.1 Требования к программному обеспечению

Входные данные: две матрицы.

Выходные данные: матрица полученная в результате умножения двух матриц.

На рисунке 2 представлена IDEF0-диаграмма, описывающая функциональную схему умножения матриц.

Рис. 2: IDEF0-диаграмма, описывающая функциональную схему умножения матриц.

3.2 Средства реализации

В данной работе используется язык программирования Python, так как ЯП позволяет написать программу за кратчайшее время. Проект выполнен в среде разработки Visual Studio Code. Многопоточное программирование было реализовано с помощью ThreadPoolExecutor. [3]

3.3 Листинг кода

В данном пункте представлен листинг кода, а именно:

- классический алгоритм умножения матриц;
- параллельная реализация классического алгоритма умножения матриц;

На листинге 1 представлен код классического алгоритма умножения матриц.

```
1        def multi(A, B):
2            if len(B) != len(A[0]):
3                print("Different dimension of the matrics")
4                return
5
6            n = len(A)
7            m = len(A[0])
8            t = len(B[0])
9
10           answer = [[0 for i in range(t)] for j in range(n)]
11           for i in range(n):
12               for j in range(m):
13                   for k in range(t):
14                       answer[i][k] += A[i][j] * B[j][k]
15           return answer
16
```

Листинг 1: Классический алгоритм умножения матриц

На листинге 2 представлен код распараллеленный алгоритм классического алгоритма умножения матриц .

```
        async def get_row(row, tmp):
            return [sum(starmap(mul,zip(row,column))) for column in tmp]

        async def multi_async(A, B):
            tmp = tuple(zip(*B))

            result = await asyncio.gather(*[get_row(row,tmp) for row in A])
            return results
        def async_mtr(A, B):
            if len(B) != len(A[0]):
                print("Different dimension of the matrics")
                return

            result = loop.run_until_complete(multi_async(A, B))
            return result

```

Листинг 2: Параллельный алгоритм умножения матриц

3.4 Вывод

В данном разделе была представлена структура ПО и листинги кода программы.

4 Исследовательская часть

В данном разделе будет проведен эксперимент и сравнительный анализ.

4.1 Системные характеристики

Характеристики компьютера на котором проводился замер времени сортировки массива:

1. операционная система - Windows 10;
2. процессор - Intel(R) Core(TM) i7-10510U CPU @1.80GHz 2.30GHz;
3. оперативная память - 16 ГБ.
4. количество ядер - 4
5. количество логических процессов - 8

4.2 Постановка эксперимента

В рамках данного проекта были проведены эксперименты, описанные ниже:

1. Сравнение времени работы алгоритмов при четном и нечетном размере матрицы для разного количества потоков.

4.3 Сравнительный анализ на основе замеров времени работы алгоритмов

Был проведен замер времени работы каждого из алгоритмов.

На рисунке 3 показаны результаты первого эксперимента, который производится для лучшего случая на квадратных матрицах размером от 100 x 100 до 1000 x 1000 с шагом 100. Количество потоков в данном эксперименте было равно 10 и 40. Ниже приведена полученная диаграмма:

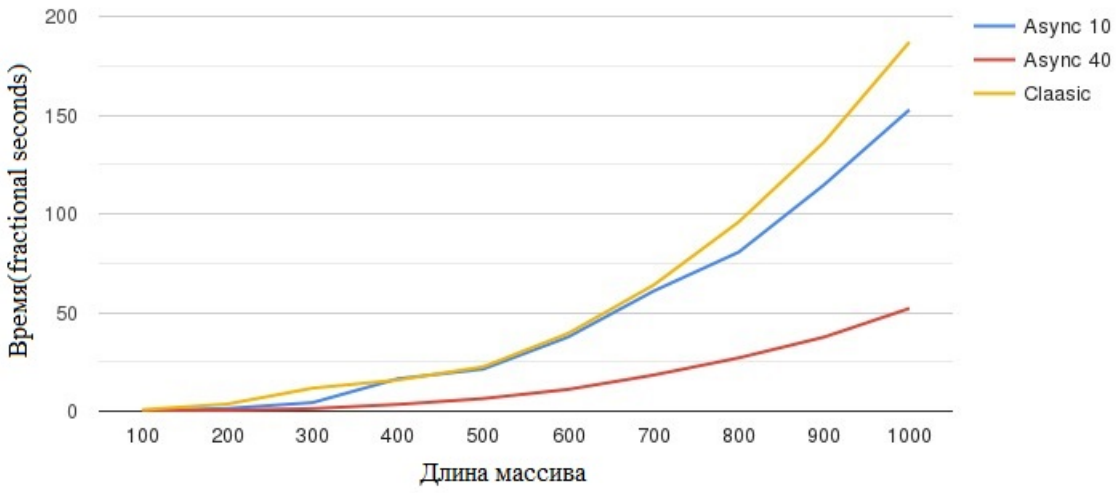

Рис. 3: Сравнение времени работы алгоритмов при четном размере матрицы

На рисунке 4 показаны результаты второго эксперимента, который производится для худшего случая, когда поданы квадратные матрицы с нечетными размерами от 101 x 101 до 1001 x 1001 с шагом 100. Количество потоков в данном эксперименте было равно 10 и 40. Ниже приведена полученная диаграмма:

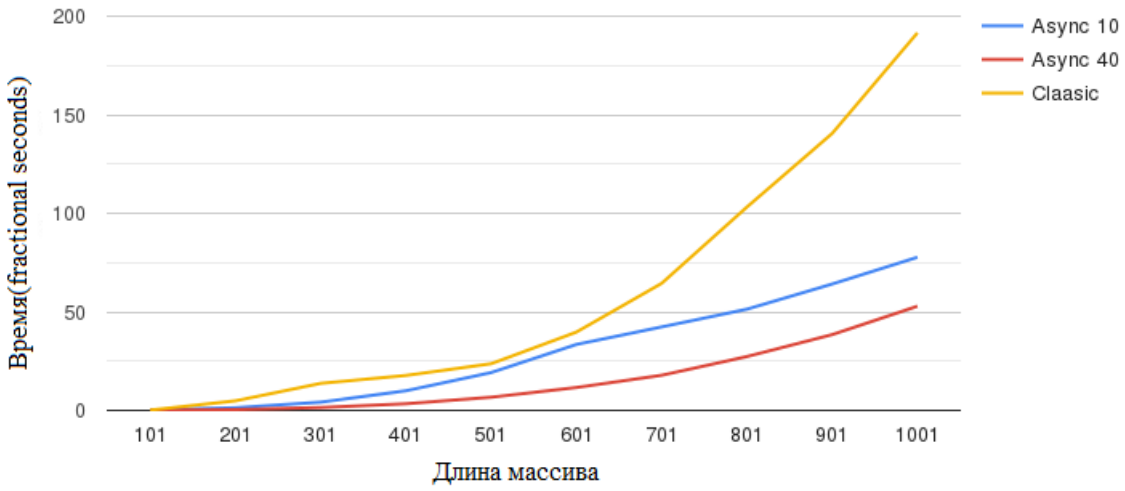

Рис. 4: Сравнение времени работы алгоритмов при нечетном размере матрицы

4.4 Тестирование программы

В данном разделе будут показаны результаты тестирования

Всего было реализовано 7 тестовых случаев:

1. некорректный размер матриц;
2. размер матрицы равен 1;
3. размер матрицы равен 2;
4. сравнение работы стандартной реализации с параллельным алгоритмом умножения на случайных значениях при четном и нечетном размерах матриц;

На рисунке 5 показаны результаты тестирования

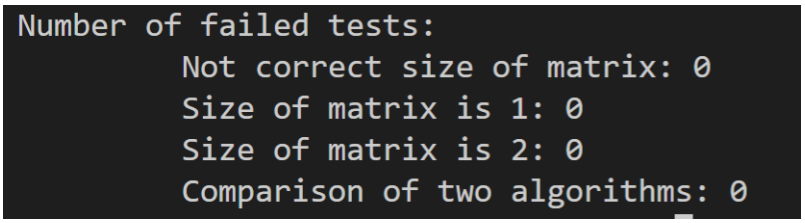

Рис. 5: Результаты тестирования

4.5 Вывод

По результатам тестирования все рассматриваемые алгоритмы были реализованы верно. По проведенному анализу можно сделать вывод что классический алгоритм умножения матриц при четном и нечетном размере матриц показал худшее время по сравнению с параллельным алгоритмом. При четном размере матриц, параллельный алгоритм с количеством пулов равным 10, показал результаты близкие к классическому алгоритму. А вот параллельный алгоритм с пулов равным 40 показал лучший результат как при четном так и при нечетном размере матриц.

Заключение

В ходе работы были изучены алгоритмы классического умножения матриц, стандартный и параллельный. Было сделано сравнение этих алгоритмов. В ходе сравнения было определо то, что самым быстрым оказался параллельный алгоритм при количестве пулов равным 40. В то время как классический алгоритм показал худшее время. Также можно выделить то, что дальнейшее увеличение пулов не даст значимую разницу во времени. Также был реализован программный код продукта.

Список литературы

[1] Ермолаев Игорь. *Умножение матриц: эффективная реализация шаг за шагом [ЭЛ. РЕСУРС] Режим доступа:* URL: `https://habr.com/ru/post/359272/`. (дата обращения: 20.10.2020).

[2] Интуит. *Введение в технологии параллельного программирования[ЭЛ. РЕСУРС] Режим доступа:* URL: `https://intuit.ru/studies/courses/4447/983/lecture/14925`. (дата обращения: 22.11.2020).

[3] Гвидо ван Россум. *Python documentation[ЭЛ. РЕСУРС] Режим доступа:* URL: `https://docs.python.org/3/library/concurrent.futures.html`. (дата обращения: 22.11.2020).