|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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»\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Рубежный контроль №1*

*По предмету: «Анализ алгоритмов»*

**Алгоритмы параллельного и асинхронного умножения матриц**

Студент: Гасанзаде М.А.,

Группа: ИУ7-56Б

Москва, 2019 г.

Оглавление

[Введение 3](#_Toc27105648)

1. [1Аналитическая часть 4](#_Toc27105649)

[1.1 Описание алгоритмов 4](#_Toc27105650)

[2 Технологическая часть 5](#_Toc27105651)

[2.1 Требования к программному обеспечению 5](#_Toc27105652)

[2.2 Средства реализации 5](#_Toc27105653)

[2.3 Листинг кода 5](#_Toc27105654)

[2.4 Описание тестирования 7](#_Toc27105656)

[3. Экспериментальная часть. 9](#_Toc27105657)

[3.1 Пример работы программы 9](#_Toc27105658)

[3.2 Сравнительный анализ алгоритмов 9](#_Toc27105659)

[3.3 Сравнительный анализ на материале экспериментальных данных 10](#_Toc27105661)

[Заключение 12](#_Toc27105663)

[Список литературы 1](#_Toc27105663)3

# Введение

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

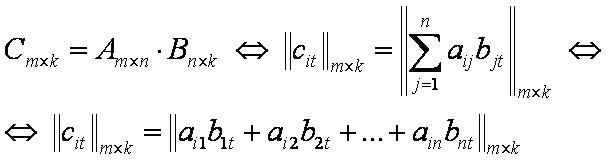
**Асинхронные вычисления –** это те вычисления, которые возникают независимо от основного потока выполнения программы. Асинхронные вычисления —  это действия обработки ввода/вывода, выполненные в неблокирующем режиме, позволяющие продолжить обработку других задач, не ожидая завершения передачи.

**Целью данной** лабораторной работы является изучение данных алгоритмов и оценка этих алгоритмов по затратам времени и памяти.

# Аналитическая часть

В данном разделе будут представлены описания алгоритмов умножения матриц.

# 1.1 Описание алгоритмов

**Произведение двух матриц** – это матрица с элементами, равными сумме произведений соответствующих элементов строк первой матрицы-сомножителя и элементов столбцов второй матрицы-сомножителя.  


Где, **m** – число строк первой матрицы-сомножителя и матрицы-произведения, **n** – число столбцов первой матрицы-сомножителя и число строк второй матрицы-сомножителя, **k** – число столбцов второй матрицы-сомножителя и матрицы-произведения, **m\*n** – размерность первой матрицы-сомножителя, **n\*k** – размерность второй матрицы-сомножителя, **m\*k** – размерность матрицы-произведения, **aij** – элемент матрицы **A**, лежащий на пересечении **i**-ой строки и **j**-ого столбца матрицы, **bjt** – элемент матрицы **B**, лежащий на пересечении **j**-ой строки и **t**-ого столбца матрицы, **cit** – элемент матрицы **C**, лежащий на пересечении **i**-ой строки и **t**-ого столбца матрицы.

* 1. **Вывод:**

В данном разделе была рассмотрена стандартная формула умножения матриц.

## Технологическая часть

В данном разделе будут приведены требования к программному обеспечению, средства реализации, листинг кода и примеры тестирования.

### Требования к программному обеспечению

На вход подаются матрицы строки, на выходе необходимо получить матрицу и 3 результата, выдаваемых матричными реализациями обоих алгоритмов и рекурсивной реализации алгоритма Дамерау-Левенштейна.

Требуется замерить время работы каждой реализации.

### Средства реализации

В качестве языка программирования был выбран Python в связи с его широким функционалом и огромнейшим набором библиотек, а также из-за привычного для меня синтаксиса. Среда разработки - стандартная IDLE Python. Время работы программы замеряется с помощью библиотеки time[1] и logging[2]. На *листинге 1* представлен код использования библиотеки logging.

**import** logging

logging**.**basicConfig**(**format**=**"[%(thread)-5d]%(asctime)s: %(message)s"**)**

logger **=** logging**.**getLogger**(**'async'**)**

logger**.**setLevel**(**logging**.**INFO**)**

logger**.**info**(**"Completed in {} seconds"**.**format**(**time**.**time**()** **-** start\_time**))**

*Листинг 1. Использование logging*

### Листинг кода

Листинг кода был представлен на *листингах* *2, 3, 4, 5.*

**def** random\_matrix**(**n**,** m**):**

**return** **[[**randint**(**0**,** 100**)** **for** i **in** range**(**m**)]** **for** j **in** range**(**n**)]**

matrix\_a **=** random\_matrix**(**3**,** 3**)**

matrix\_b **=** random\_matrix**(**3**,** 3**)**

*Листинг 2. Генерация рандомной матрицы.*

**def** multi**(**A**,** B**):**

**if** len**(**B**)** **!=** len**(**A**[**0**]):**

**print(**"Different dimension of the matrics"**)**

**return**

n **=** len**(**A**)**

m **=** len**(**A**[**0**])**

t **=** len**(**B**[**0**])**

answer **=** **[[**0 **for** i **in** range**(**t**)]** **for** j **in** range**(**n**)]**

**for** i **in** range**(**n**):**

**for** j **in** range**(**m**):**

**for** k **in** range**(**t**):**

answer**[**i**][**k**]** **+=** A**[**i**][**j**]** **\*** B**[**j**][**k**]**

**return** answer

**async** **def** multi\_async**(**A**,** B**):**

tmp **=** tuple**(**zip**(\***B**))**

results **=** **await** asyncio**.**gather**(\*[**get\_new\_elem**(**row**,** tmp**)** **for** row **in** A**])**

**return** results

*Листинг 4. Функция умножения матрицы при параллельной реализации.*

**def** multi\_matrices**(**first**,** second**,** third**):**

res **=** **[[**0**,** 0**,** 0**],** **[**0**,** 0**,** 0**],** **[**0**,** 0**,** 0**]]**

**for** i **in** range**(**len**(**first**)):**

**for** j **in** range**(**len**(**first**)):**

res**[**i**][**j**]** **=** first**[**i**][**j**]** **+** second**[**i**][**j**]** **+** third**[**i**][**j**]**

**return** res

*Листинг 3. Умножение матрицы при асинхронной реализации.*

**class** **Thread1(**threading**.**Thread**):**

*@profile*

**def** \_\_init\_\_**(**self**,** id**,** name**):**

threading**.**Thread**.**\_\_init\_\_**(**self**)**

self**.**id **=** id

self**.**name **=** name

*@profile*

**def** run**(**self**):**

**print** **(**"Starting " **+** self**.**name **+** "\n"**)**

vector\_3x1a **=** **[**matrix\_a**[**i**][**0**]** **for** i **in** range**(**len**(**matrix\_a**))]**

vector\_1x3b **=** **[**matrix\_b**[**0**][**j**]** **for** j **in** range**(**len**(**matrix\_b**[**0**]))]**

matrix\_c\_aux1**[**0**][**0**]** **=** vector\_3x1a**[**0**]** **\*** vector\_1x3b**[**0**]**

matrix\_c\_aux1**[**0**][**1**]** **=** vector\_3x1a**[**0**]** **\*** vector\_1x3b**[**1**]**

matrix\_c\_aux1**[**0**][**2**]** **=** vector\_3x1a**[**0**]** **\*** vector\_1x3b**[**2**]**

matrix\_c\_aux1**[**1**][**0**]** **=** vector\_3x1a**[**1**]** **\*** vector\_1x3b**[**0**]**

matrix\_c\_aux1**[**1**][**1**]** **=** vector\_3x1a**[**1**]** **\*** vector\_1x3b**[**1**]**

matrix\_c\_aux1**[**1**][**2**]** **=** vector\_3x1a**[**1**]** **\*** vector\_1x3b**[**2**]**

matrix\_c\_aux1**[**2**][**0**]** **=** vector\_3x1a**[**2**]** **\*** vector\_1x3b**[**0**]**

matrix\_c\_aux1**[**2**][**1**]** **=** vector\_3x1a**[**2**]** **\*** vector\_1x3b**[**1**]**

matrix\_c\_aux1**[**2**][**2**]** **=** vector\_3x1a**[**2**]** **\*** vector\_1x3b**[**2**]**

**print** **(**"End " **+** self**.**name **+** "\n"**)**

### 

*Листинг 5. Реализация потока в параллельной реализации*

### Описание тестирования

Так как для максимального быстродействия в параллельном умножении были созданы 3 потока для каждой матрицы, мы будем рассматривать рандомные матрицы 3x3. Таблицы с результатами были представлены на *рис.* 1, 2.

*Рисунок 1 — Данные тестов при генерации значений матрицы от 0 до 100*

|  |  |
| --- | --- |
| **Async** | **Parallel** |
| **0.0019991397857666016** | **0.028941869735717773** |
| **0.001999378204345703** | **0.03298020362854004** |
| **0.0009829998016357422** | **0.07339262962341309** |
| **0.000982522964477539** | **0.07525110244750977** |
| **0.0009829998016357422** | **0.0809319019317627** |

*Рисунок 2 – Данные тестов при генерации значений матрицы от 0 до 1000*

|  |  |
| --- | --- |
| **Async**  **Время в секундах** | **Parallel**  **Время в секундах** |
| **0.0019989013671875** | **0.025501251220703125** |
| **0.0009992122650146484** | **0.03381609916687012** |
| **0.00099945068359375** | **0.0721273422241211** |
| **0.0019986629486083984** | **0.07727336883544922** |
| **0.001979351043701172** | **0.07005810737609863** |
| **0.0009834766387939453** | **0.06937527656555176** |
| **0.0019989013671875** | **0.06998252868652344** |

Все тесты пройдены успешно.

* 1. **Вывод**

В данном разделе мы рассмотрели листинг кода, а также убедились в безошибочной работе программы.

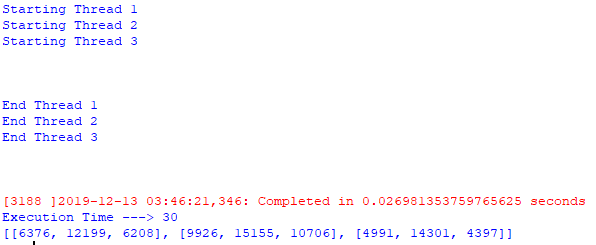
1. **Экспериментальная часть.**

В данном разделе будут рассмотрены примеры работ программы.

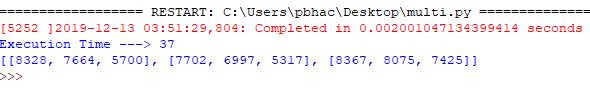
Память была замерена с помощью библиотеки memory\_profiler[4] в 32х битном Python IDLE.

### Пример работы программы

На рисунках 3, 4 приведены изображения внешнего вида интерфейса программы во время его работы



*Рисунок 4, пример работы программы параллельного умножения.*

**

*Рисунок 5, пример работы программы асинхронного умножения.*

* 1. **Сравнительный анализ алгоритмов**

На примере малоразмерных матриц было показано, что алгоритм параллельного умножении матриц, сильно проигрывает по времени и затрачиваемой памяти, асинхронному алгоритму.

### Сравнительный анализ на материале экспериментальных данных

# Алгоритмы были протестированы по скорости работы и используемой памяти.

На рисунках 1 и 2.

*Рисунок 1- Использование памяти.*

*Рисунок 2 – Замер времени*

* 1. **Вывод**

В данном разделе был представлен эксперимент по замеру времени и памяти выполнения каждого алгоритма. По итогам замеров алгоритм параллельного умножения матрицы показал себя хуже чем асинхронный во всех замерах.

# Заключение

В данной работе были реализованы и протестированы два современных методов умножения матриц. Проведён сравнительный анализ обоих реализаций.

Асинхронная реализация показала себя намного лучше из-за особенностей работы Python3, и малоразмерности матриц.[3]

Параллельная реализация эффективна при применении к большим матрицам.

**Список литературы**

1. time — Time access and conversions // Python URL: https://docs.python.org/3/library/time.html
2. logging – <https://docs.python.org/3/library/logging.html>
3. Использование асинхронных вычислений: <http://ikit.sfu-kras.ru/files/ikit/03_Asinhronnye_spiski.pdf>
4. Замер памяти в Python3: URL: <https://pypi.org/project/memory-profiler/>
5. Параллельные алгоритмы умножения матриц: [Shortulr.wizardmh](1.%09http:/ssd.sscc.ru/ru/content/%D0%BF%D0%B0%D1%80%D0%B0%D0%BB%D0%BB%D0%B5%D0%BB%D1%8C%D0%BD%D1%8B%D0%B5-%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D1%8B-%D1%83%D0%BC%D0%BD%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D1%8F-%D0%BC%D0%B0%D1%82%D1%80%D0%B8%D1%86)