**МИНОБРНАУКИ РОССИИ**
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**
**Кафедра МО ЭВМ**

# ОТЧЕТ

**по лабораторной работе №4**
**по дисциплине «Параллельные алгоритмы»**
**Тема: Параллельное умножение матриц.**

Студент гр. 9303 ____________________ Камакин Д.В.

Преподаватель ____________________ Сергеева Е.И.

Санкт-Петербург
2022

**Цель работы.**

Реализовать алгоритмы параллельного умножения матриц.

**Задание.**

4.1 Реализовать параллельный алгоритм умножения матриц.
Исследовать масштабируемость выполненной реализации.

4.2 Реализовать параллельный алгоритм “быстрого” умножения матриц (Штрассена или его модификации).

Проверить, что результаты вычислений реализаций 4.1 и 4.2 совпадают.

Сравнить производительность с реализацией 4.1 на больших размерностях данных (порядка 10^4 – 10^6)

**Выполнение работы.**

При помощи набора (пула) потоков были реализованы два алгоритма параллельного умножения матриц.

Первый алгоритм (Наивный) рассматривает матрицу как одномерный массив и заключается в разбиении набора данных на части, каждая из которая оборачивается в объект Runnable, являющийся задачей нашего пула потоков. Поскольку части независимы друг от друга, то и нет необходимости синхронизации самого алгоритма, лишь в очереди задач. После каждой задачи поток увеличивает счётчик и матрица считается умноженной, когда значение этой переменной будет равно количеству созданных задач.

Второй алгоритм (Штрассена) в оригинальном виде является рекурсивным. Для распараллеливания также воспользуемся пулом потоков и постараемся разбить алгоритм на минимальные независимые части, чтобы уменьшить конкуренцию между потоками и увеличить пропускную способность. Кратко алгоритм описывается так (для понимания опустим математические детали):

1. Разбить матрицу на 7 частей.
2. Рекурсивно для каждой из 7 частей выполнить п.1, п.2, а затем совместить в единый результат.

Из описания извлечём два основных типа задач: деление и совмещение. Первая будет реализовывать ту самую рекурсию, но для распараллеливания придётся её “развернуть”. Каждая задача будет представлять свой уровень рекурсии и в ней поток делит матрицу на части, после чего для каждой части формирует задачу “деление” и кладёт в очередь. Таким образом для вычисления не требуется ждать другие потоки и уровни могут рассчитываться независимо друг от друга. Представить можно так: изначальная матрица 1024x1024 потоком A делится на части размером 512, каждая из частей добавляется в очередь, после чего потенциально поток B (однако может и снова поток A) делит часть на другие, но уже размером 256 и так далее.

Однако даже из этого описания может возникнуть вопрос: как понять, что части просчитаны и пора их “совмещать”? Действительно, мы не имеем информации даже о количестве частей на вложенных уровнях, не говоря уже о времени их выполнения. Предлагаемое решение следующее: будем считать, что части одного уровня пора совмещать, когда каждая из них будет рассчитана. Для этого достаточно отслеживать, была ли изменена матрица самой части. Для этого задачу деления дополним ссылкой на функцию, которая будет выполняться после самой задачи и она будет проверять, вычислили ли мы сейчас какую-то часть и если да, и все задачи готовы, то добавит предыдущий уровень как задачу на совмещение.

Таким образом было реализовано распараллеливание алгоритма Штрассена, в котором задачам нет необходимости ждать друг друга и каждая может выполняться независимо от другой.

Сравним результаты работы алгоритмов:

```
512 512
11877 11699 12065 12101 13645 12216 14032 12259 12926
12765 13185 12728 12607 12260 12419 12254 13031 12687
12861 12380 12840 12535 13377 12402 12258 12734 12219
12458 12485 12461 12716 12837 12402 12447 12444 12715
12431 12483 12685 12821 12231 12699 12504 13092 12258
12034 12292 14032 12584 13021 12929 12602 12365 12850
12747 12795 12700 13208 11944 11975 12777 12678 13131
12882 12800 12174 12525 13077 12286 14105 12176 13282
12407 12462 11685 12619 12347 12735 13245 13396 11357
12649 12620 12888 12920 12900 12304 12623 12236 12291
12658 13133 12468 13168 13878 12390 12152 12901 12421
13176 12372 13094 12497 12644 12320 12228 12428 13300
12976 13130 12748 12207 13647 13077 13048 12857 12415
12271 12103 12575 12345 12396 12425 12820 12696 12386
12026 12838 12381 12379 13355 12285 12437 12676 11878
12961 12595 13083 13125 12952 13055 13354 12730 12753
12817 12300 12450 12777 13695 12556 14460 12644 13348
12832 12943 13234 12993 12713 12773 12239 13621 13466
13516 13431 12760 12660 13263 13432 12180 13473 12396
12181 13105 12682 12731 13007 12927 12913 12541 13402
```

Рисунок 1 - Результат работы первого алгоритма

```
512 512
11877 11699 12065 12101 13645 12216 14032 12259 12926
12765 13185 12728 12607 12260 12419 12254 13031 12687
12861 12380 12840 12535 13377 12402 12258 12734 12219
12458 12485 12461 12716 12837 12402 12447 12444 12715
12431 12483 12685 12821 12231 12699 12504 13092 12258
12034 12292 14032 12584 13021 12929 12602 12365 12850
12747 12795 12700 13208 11944 11975 12777 12678 13131
12882 12800 12174 12525 13077 12286 14105 12176 13282
12407 12462 11685 12619 12347 12735 13245 13396 11357
12649 12620 12888 12920 12900 12304 12623 12236 12291
12658 13133 12468 13168 13878 12390 12152 12901 12421
13176 12372 13094 12497 12644 12320 12228 12428 13300
12976 13130 12748 12207 13647 13077 13048 12857 12415
12271 12103 12575 12345 12396 12425 12820 12696 12386
12026 12838 12381 12379 13355 12285 12437 12676 11878
12961 12595 13083 13125 12952 13055 13354 12730 12753
12817 12300 12450 12777 13695 12556 14460 12644 13348
12832 12943 13234 12993 12713 12773 12239 13621 13466
13516 13431 12760 12660 13263 13432 12180 13473 12396
12181 13105 12682 12731 13007 12927 12913 12541 13402
```

Рисунок 2 - Результат работы второго алгоритма

Видно, что алгоритмы выдают одинаковый верный результат.

| Количество потоков | Размер квадратной матрицы | Алгоритм | Время выполнения (мс) |
|---|---|---|---|
| 1 | 64 | Наивный | 2 |
| 1 | 64 | Штрассена | 2 |
| 1 | 512 | Наивный | 1522 |
| 1 | 512 | Штрассена | 1141 |
| 1 | 1024 | Наивный | 12035 |
| 1 | 1024 | Штрассена | 8408 |
| 1 | 2048 | Наивный | 122423 |
| 1 | 2048 | Штрассена | 60093 |
| 2 | 64 | Наивный | 2 |
| 2 | 64 | Штрассена | 2 |
| 2 | 512 | Наивный | 838 |
| 2 | 512 | Штрассена | 658 |
| 2 | 1024 | Наивный | 6911 |
| 2 | 1024 | Штрассена | 4457 |
| 2 | 2048 | Наивный | 68329 |
| 2 | 2048 | Штрассена | 33825 |
| 3 | 64 | Наивный | 2 |
| 3 | 64 | Штрассена | 2 |
| 3 | 512 | Наивный | 701 |
| 3 | 512 | Штрассена | 430 |
| 3 | 1024 | Наивный | 5421 |
| 3 | 1024 | Штрассена | 3430 |
| 3 | 2048 | Наивный | 57040 |
| 3 | 2048 | Штрассена | 27763 |

Таблица 1 - Сравнение реализованных алгоритмов

**Выводы.**

На языке программирования C++ были реализованы параллельные алгоритмы умножения матриц. В результате сравнения видно, что разработанное решение успешно масштабируется при увеличении размеров матрицы и потоков (время выполнения меньше при большем количестве потоков), а также предложенная модификация алгоритма Штрассена в среднем в два раза быстрее наивного подхода.