Национальный исследовательский ядерный университет «МИФИ» (Московский Инженерно–Физический Институт)

Кафедра №42 «Криптология и кибербезопасность»

Лабораторная работа №2:

«Введение в docker-compose и nginx»

Тимин Александр Б21-515 (2023г.)

Алгоритм

Без балансировщика нагрузки

Два сервера **producer** генерируют по **N×N** псевдослучайных чисел и отправляют их по сети на сервер-обработчик **consumer**, который собирает из полученных чисел две квадратные матрицы и перемножает их, замеряя время работы алгоритма.

С балансировщиком нагрузки

Два сервера **producer** генерируют по **N×N** псевдослучайных чисел и отправляют их по сети на сервер **agregator**. Сервер **agregator** собирает из полученных чисел две квадратные матрицы, разделяет их (в соответствии с алгоритмом блочного умножения матриц) на блоки и отправляет пары блоков на сервер балансировки **balancer**. Сервер балансировки распределяет полученные пары блоков по серверам **consumer**. Каждый из серверов **consumer** принимает пары блоков, находит их произведение и отправляет его на сервер **island**. Сервер **island** принимает результаты блочного умножения и распределяет их по группам для сложения (в соответствии с алгоритмом блочного умножения матриц) и отправляет полученные группы на сервер балансировки **balancer**. Сервер балансировки распределяет полученные группы блоков по серверам **consumer**. Каждый из серверов **consumer** принимает группы блоков, находит их сумму и отправляет ее на сервер **collector**. Сервер **collector**, принимая блоки, (в соответствии с алгоритмом блочного умножения матриц) формирует из них результирующую матрицу.

Docker-compose

- version: ‘3’ – указывает версию Docker Compose, используемую в файле
- services: – начало секции, где определяются сервисы (контейнеры) для запуска
- producer1: – имя сервиса (контейнера) producer1
- build: – определяет настройки для сборки контейнера
- context: ./producer – указывает путь к директории с Dockerfile для сборки контейнера

- no_cache: true – отключает кэширование при сборке контейнера producer1
- environment: – задает переменные окружения, которые будут загружены в контейнер producer1
- env_file: – указывает файлы с переменными окружения, которые будут загружены в контейнер producer1
- networks: – определяет сети, к которым будет присоединен контейнер producer1
- ports: – определяет открытые порты для сервиса (контейнера)

Dockerfile

- FROM alpine:latest - используем основой latest версию образа alpine
- RUN - команды, вызываемые в процессе сборки образа
- WORKDIR /app - задание директории /app в качестве рабочей директории
- CMD - команда, вызываемая в процессе запуска контейнера

Заключение

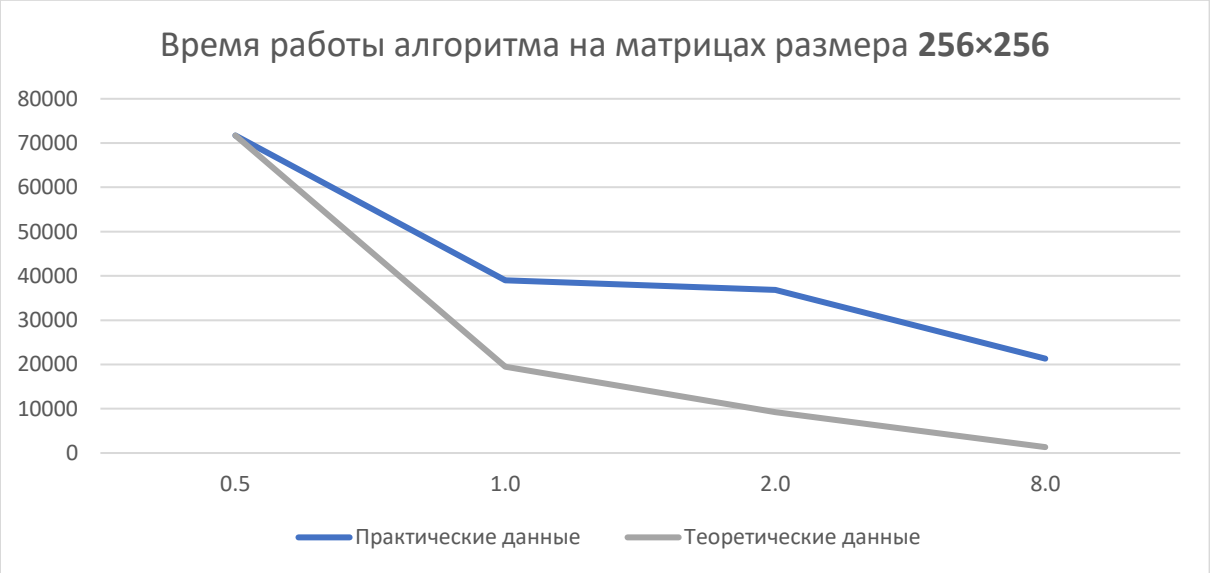
Был разработан алгоритм умножения матриц.

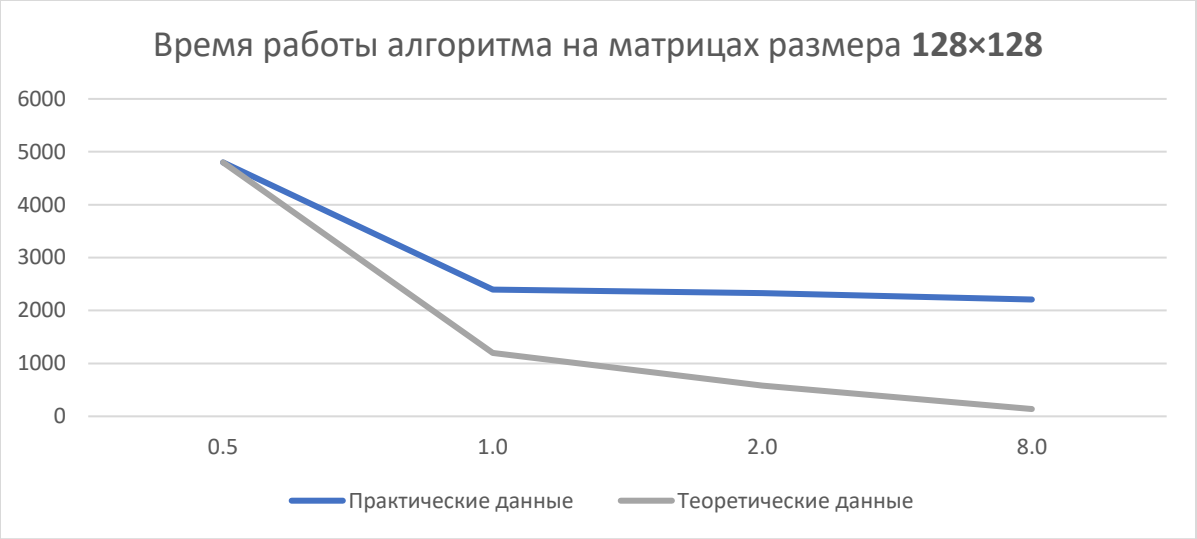
Были написаны небходимые Dockerfile для серверов, docker-compose.yml для сборки приложения и необходимые программы для backend серверов. С помощью них был собран образ многоконтейнерного приложения, который при запуске считал произведение квадратных матриц и выводил время работы алгоритма.

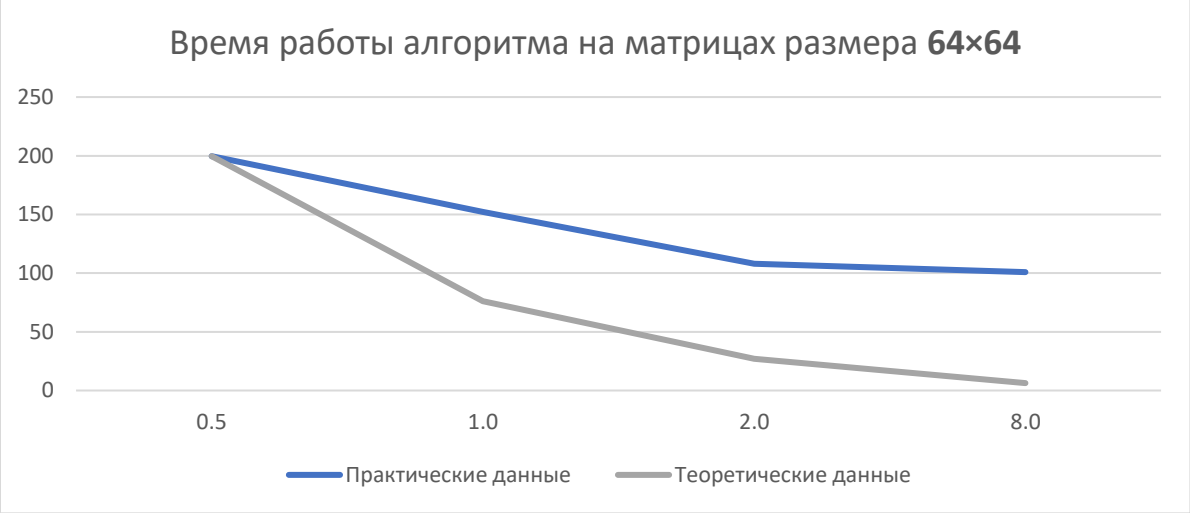
Для различных размеров матриц и ограничений производительности были получены временые характеристики работы алгоритма без балансировщика нагрузки, средние значения которых приведены в таблице.

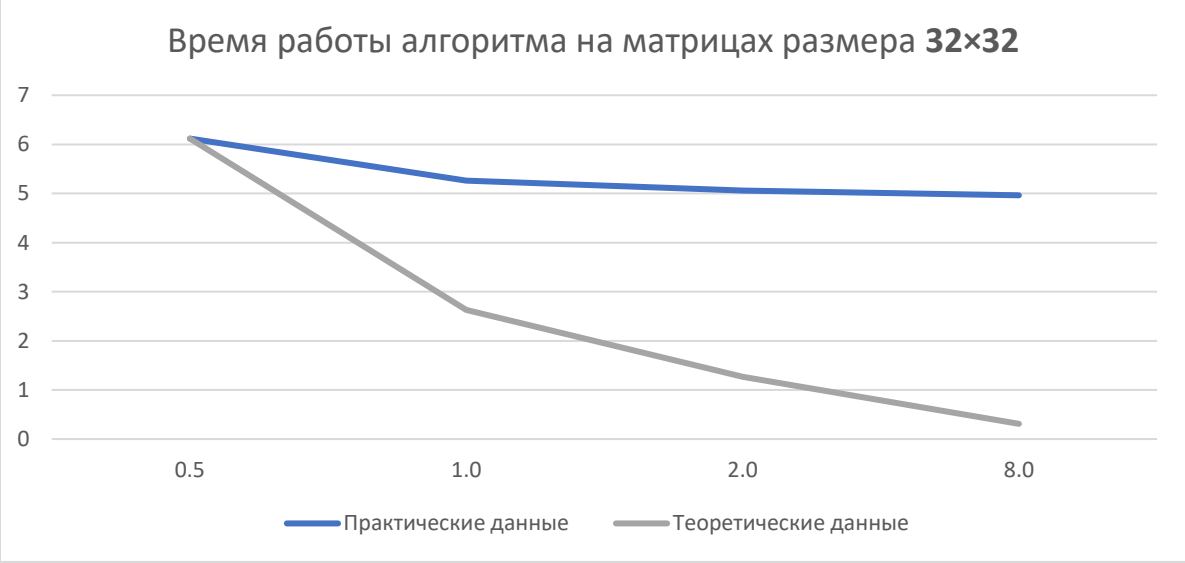
#	cpus	time (ms, pr)	Msize	time (ms, th)
1	0.5	71712.500	256	71712.500
2	1.0	38980.700	256	19490.350
3	2.0	36820.000	256	9205.000
4	8.0	21313.500	256	1332.094
5	0.5	4799.950	128	4799.950
6	1.0	2394.930	128	1197.465
7	2.0	2328.690	128	582.173
8	8.0	2208.030	128	138.002
9	0.5	199.622	64	199.622
10	1.0	152.228	64	76.114
11	2.0	108.059	64	27.015
12	8.0	100.849	64	6.303
13	0.5	6.118	32	6.118
14	1.0	5.259	32	2.629
15	2.0	5.060	32	1.265
16	8.0	4.963	32	0.310

По полученным временным характеристикам были построены графики для оценки времени исполнения алгоритма.




Время работы алгоритма на матрицах размера 128×128
6000
5000
4000
3000
2000
1000
0
0.5
1.0
2.0
8.0
Практические данные
Теоретические данные


Время работы алгоритма на матрицах размера 64×64
250
200
150
100
50
0
0.5
1.0
2.0
8.0
Практические данные
Теоретические данные


Время работы алгоритма на матрицах размера 32×32
7
6
5
4
3
2
1
0
0.5
1.0
2.0
8.0
Практические данные
Теоретические данные

По полученным данным видно наличие прироста производительности, однако он гораздо меньше, чем в классической многопроцессорной разработке. Это может быть связано с дополнительной нагрузкой на систему от контейнерного окружения.

Приложение

- **https://github.com/KATEHOK/parvpo_2** - репозиторий со всеми материалами
- **https://github.com/KATEHOK/parvpo_2/blob/main/vertical.rar** - архив с материалами алгоритма без балансировки нагрузки
- **https://github.com/KATEHOK/parvpo_2/blob/main/docker-compose.yaml** - docker-compose файл для алгоритма с балансировкой нагрузки
- **https://github.com/KATEHOK/parvpo_2/tree/main/producer** - материалы для серверов **producer** для алгоритма с балансировкой нагрузки
- **https://github.com/KATEHOK/parvpo_2/tree/main/agregator** - материалы для сервера **agregator** для алгоритма с балансировкой нагрузки
- **https://github.com/KATEHOK/parvpo_2/tree/main/balancer** - материалы для сервера **balancer** для алгоритма с балансировкой нагрузки
- **https://github.com/KATEHOK/parvpo_2/tree/main/consumer** - материалы для серверов **consumer** для алгоритма с балансировкой нагрузки
- **https://github.com/KATEHOK/parvpo_2/tree/main/island** - материалы для сервера **island** для алгоритма с балансировкой нагрузки
- **https://github.com/KATEHOK/parvpo_2/tree/main/collector** - материалы для сервера **collector** для алгоритма с балансировкой нагрузки
- **https://github.com/KATEHOK/parvpo_2/tree/main/report** - директория с таблицей и отчетом