Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«Пермский государственный исследовательский университет»

ИНСТИТУТ КОМПЬЮТЕРНЫХ НАУК И ТЕХНОЛОГИЙ

НАПРАВЛЕНИЕ ПОДГОТОВКИ **01.03.02 Прикладная математика и информатика**

О Т Ч Е Т

По лабораторной работе № 3

Название: Введение в MPI. Операции с массивами. Пузырьковая сортировка

Дисциплина: Параллельные вычислительные системы

Студент ИТ-3,4 (Группа) _____ (Подпись, дата) Н.М. Коньшин (И.О. Фамилия)

Преподаватель _____ (Подпись, дата) А.С. Белозеров (И.О. Фамилия)

Пермь, 2025

Введение

Цель работы: приобретение знаний, умений и навыков в области технологии параллельного программирования средствами библиотеки MPI.

Задание работы

1. Изучить основы технологии параллельного программирования средствами библиотеки MPI.
2. Используя средства библиотеки MPI, научиться решать задачи сложения (и других арифметических операций) элементов.
3. Используя средства библиотеки MPI, научиться реализовывать параллельную работу пузырьковой сортировки элементов.

Основная часть

- Задания данной лабораторной работы частично пересекаются с заданиями ЛР №2. Поэтому часть скриптов и написанных программ была взята именно оттуда. Весь дальнейший код программ представлен на Github в моем репозитории.
- Скрипт на генерацию массива / матриц у нас уже готов, поэтому в очередной раз воспользуемся им для создания массива для первого задания. Только в этот раз я решил сильно увеличить его объем, чтобы протестировать работу на большем количестве данных.
- Создаем массив на 5.000.000 элементов, каждый из которых равен от 1 до 1000.

```
import random

# Генерируем массив из 5000000 случайного числа от 1 до 1000
arr = [random.randint(1, 1000) for _ in range(5000000)]

# Сохраняем массив в файл
with open("array.txt", "w") as f:
    f.write(" ".join(map(str, arr)))

print("Массив сохранён в файл array.txt")
```

Task 1

- Результат выполнения программ в локальной среде программирования

```
~/Documents/PSU/2.4sem/Параллельные вычислительные системы/Labs_git/LR3/Task1 main ± ./sequential_sum
Размер массива: 5000000 элементов
Сумма элементов: 2502714968
Время выполнения: 0.491288 секунд
~/Documents/PSU/2.4sem/Параллельные вычислительные системы/Labs_git/LR3/Task1 main ± mpirun -np 4 ./parallel_sum
=== ПАРАЛЛЕЛЬНАЯ ВЕРСИЯ (MPI) ===
Количество процессов: 4
Размер массива: 5000000 элементов
Сумма элементов: 2502714968
Время выполнения: 0.005488 секунд
```

И здесь мы сразу видим крайне впечатляющие результаты, так как ускорение от использования параллельного варианта достигает 89x.

- Результат выполнения программ на «ПГНИУ-Кеплер»

```
Successfully completed.

Resource usage summary:

    CPU time :                                   0.95 sec.
    Max Memory :                                 8 MB
    Average Memory :                             8.00 MB
    Total Requested Memory :                     -
    Delta Memory :                               -
    Max Swap :                                   -
    Max Processes :                              4
    Max Threads :                                5
    Run time :                                   10 sec.
    Turnaround time :                            7 sec.

The output (if any) follows:

Размер массива: 5000000 элементов
Сумма элементов: 2502714968
Время выполнения: 0.837184 секунд
```

```
Successfully completed.

Resource usage summary:

    CPU time :                                   5.89 sec.
    Max Memory :                                 10 MB
    Average Memory :                             10.00 MB
    Total Requested Memory :                     -
    Delta Memory :                               -
    Max Swap :                                   -
    Max Processes :                              7
    Max Threads :                                8
    Run time :                                   6 sec.
    Turnaround time :                            8 sec.

The output (if any) follows:

=== ПАРАЛЛЕЛЬНАЯ ВЕРСИЯ (MPI) ===
Количество процессов: 4
Размер массива: 5000000 элементов
Сумма элементов: 2502714968
Время выполнения: 0.010628 секунд
```

```
Successfully completed.

Resource usage summary:

    CPU time :                                   12.04 sec.
    Max Memory :                                 9 MB
    Average Memory :                             6.00 MB
    Total Requested Memory :                     -
    Delta Memory :                               -
    Max Swap :                                   -
    Max Processes :                              4
    Max Threads :                                5
    Run time :                                   5 sec.
    Turnaround time :                            9 sec.

The output (if any) follows:

=== ПАРАЛЛЕЛЬНАЯ ВЕРСИЯ (MPI) ===
Количество процессов: 8
Размер массива: 5000000 элементов
Сумма элементов: 2502714968
Время выполнения: 0.013507 секунд
```

Здесь также заметна разница в ускорении работы программы, при использовании библиотеки MPI. Самым быстрым оказался вариант при разбиении задачи на 4 процесса. Но он все равно уступает по скорости работы программы, запущенной локально.

Task 2

- Второе задание по смыслу также похоже на задачу из предыдущей лабораторной работы, только здесь нужно было реализовать не быструю сортировку, а пузырьковую. Пузырьковая сортировка - это простой алгоритм сортировки, который последовательно сравнивает соседние элементы массива и меняет их местами, если они находятся в неправильном порядке.
- Для данного задания мне пришлось уменьшить массив данных обратно до 100.001 элемента из-за времени работы программы. Так как средняя временная сложность данного алгоритма $O(n^2)$, что гораздо больше той же быстрой сортировки. Для 5 000 000 элементов пузырьковая сортировка выполнит примерно $(5\ 000\ 000)^2 / 2 = 12\ 500\ 000\ 000\ 000$ операций сравнения и обмена. Современный процессор выполняет около 10^9 операций в секунду на одном ядре. И таким образом расчетное время

составит * 12.5 триллиона операций / 1 миллиард операций в секунду = 12 500 секунд ≈ 3.5 часа.

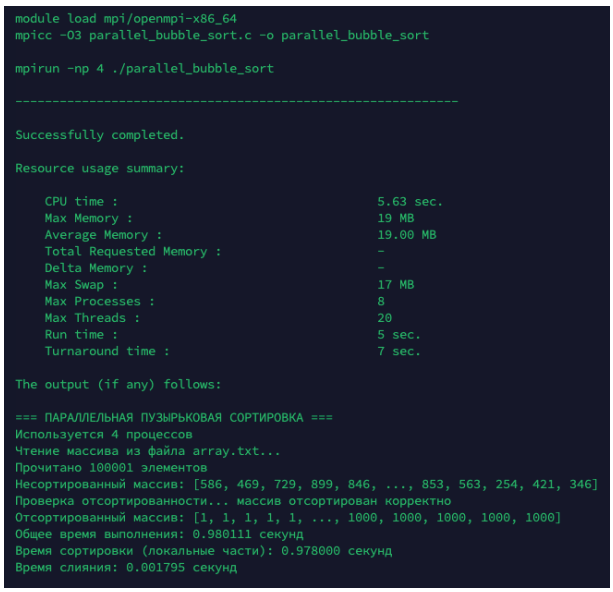
- Для примера, сравнение с быстрой сортировкой: для 5M элементов:
  O(n log n) ≈ 5M × 22 ≈ 110M операций.
  Примерное время: 0.1-0.2 секунды.
- Результат выполнения программ в локальной среде программирования

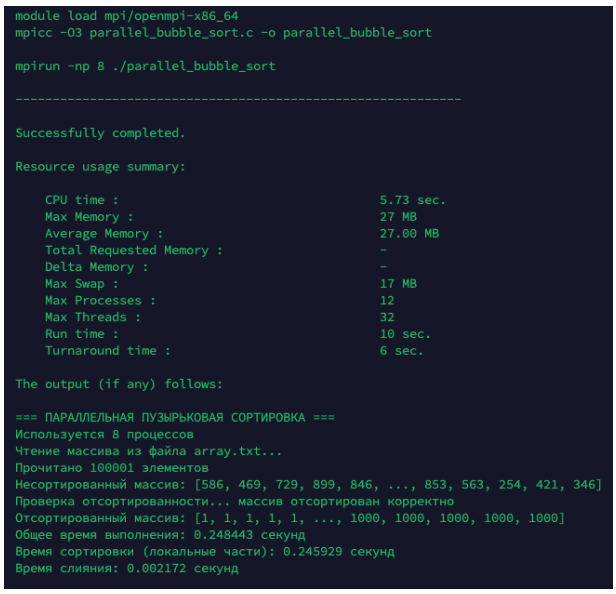
```
~/Documents/PSU/2.4sem/Параллельные вычислительные системы/Labs_git/LR3/Task2 main ± ./sequential_bubble_sort
=== ПОСЛЕДОВАТЕЛЬНАЯ ПУЗЫРЬКОВАЯ СОРТИРОВКА ===
Чтение массива из файла array.txt...
Прочитано 100001 элементов
Несортированный массив: [586, 469, 729, 899, 846, ..., 853, 563, 254, 421, 346]
Сортировка...
Проверка отсортированности... массив отсортирован корректно
Отсортированный массив: [1, 1, 1, 1, 1, ..., 1000, 1000, 1000, 1000, 1000]
Время сортировки: 8.730939 секунд
Общее время выполнения: 8.744674 секунд
~/Documents/PSU/2.4sem/Параллельные вычислительные системы/Labs_git/LR3/Task2 main ± mpirun -np 3 ./parallel_bubble_sort
=== ПАРАЛЛЕЛЬНАЯ ПУЗЫРЬКОВАЯ СОРТИРОВКА ===
Используется 3 процессов
Чтение массива из файла array.txt...
Прочитано 100001 элементов
Несортированный массив: [586, 469, 729, 899, 846, ..., 853, 563, 254, 421, 346]
Проверка отсортированности... массив отсортирован корректно
Отсортированный массив: [1, 1, 1, 1, 1, ..., 1000, 1000, 1000, 1000, 1000]
Общее время выполнения: 0.404280 секунд
Время сортировки (локальные части): 0.394391 секунд
Время слияния: 0.008525 секунд
~/Documents/PSU/2.4sem/Параллельные вычислительные системы/Labs_git/LR3/Task2 main ± mpirun -np 6 ./parallel_bubble_sort
=== ПАРАЛЛЕЛЬНАЯ ПУЗЫРЬКОВАЯ СОРТИРОВКА ===
Используется 6 процессов
Чтение массива из файла array.txt...
Прочитано 100001 элементов
Несортированный массив: [586, 469, 729, 899, 846, ..., 853, 563, 254, 421, 346]
Проверка отсортированности... массив отсортирован корректно
Отсортированный массив: [1, 1, 1, 1, 1, ..., 1000, 1000, 1000, 1000, 1000]
Общее время выполнения: 0.087282 секунд
Время сортировки (локальные части): 0.079414 секунд
Время слияния: 0.006551 секунд
~/Documents/PSU/2.4sem/Параллельные вычислительные системы/Labs_git/LR3/Task2 main ±
```

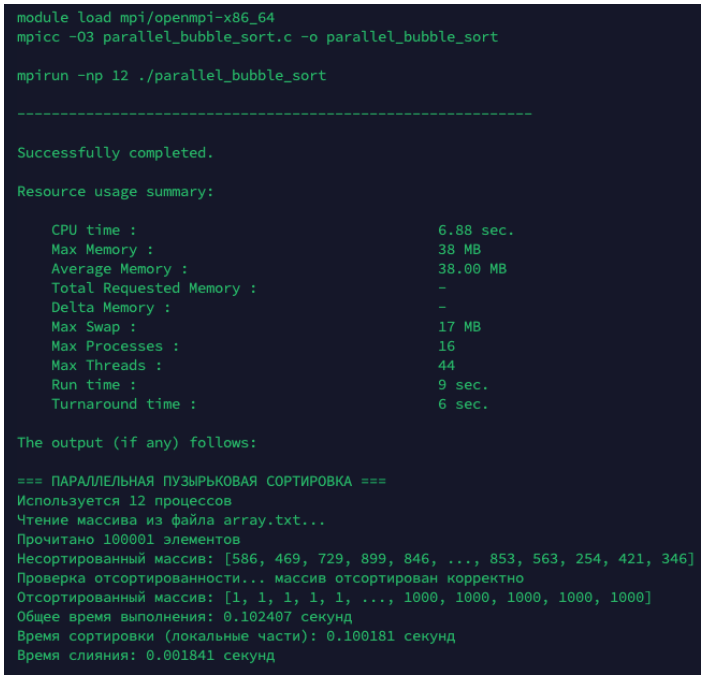
Здесь также видна огромная разница по скорости работы программ. Параллельный вариант на 3 процесса быстрее последовательного варианта программы в 21 раз, а при использовании 6 процессов скорость работы становится больше еще на 130%.

- Результат выполнения программ на «ПГНИУ-Кеплер»

```
Successfully completed.

Resource usage summary:

    CPU time :                                   16.55 sec.
    Max Memory :                                 9 MB
    Average Memory :                             8.20 MB
    Total Requested Memory :                     -
    Delta Memory :                               -
    Max Swap :                                   2 MB
    Max Processes :                              6
    Max Threads :                                7
    Run time :                                   19 sec.
    Turnaround time :                            20 sec.

The output (if any) follows:

=== ПОСЛЕДОВАТЕЛЬНАЯ ПУЗЫРЬКОВАЯ СОРТИРОВКА ===
Чтение массива из файла array.txt...
Прочитано 100001 элементов
Несортированный массив: [586, 469, 729, 899, 846, ..., 853, 563, 254, 421, 346]
Сортировка...
Проверка отсортированности... массив отсортирован корректно
Отсортированный массив: [1, 1, 1, 1, 1, ..., 1000, 1000, 1000, 1000, 1000]
Время сортировки: 16.393916 секунд
Общее время выполнения: 16.411067 секунд
```



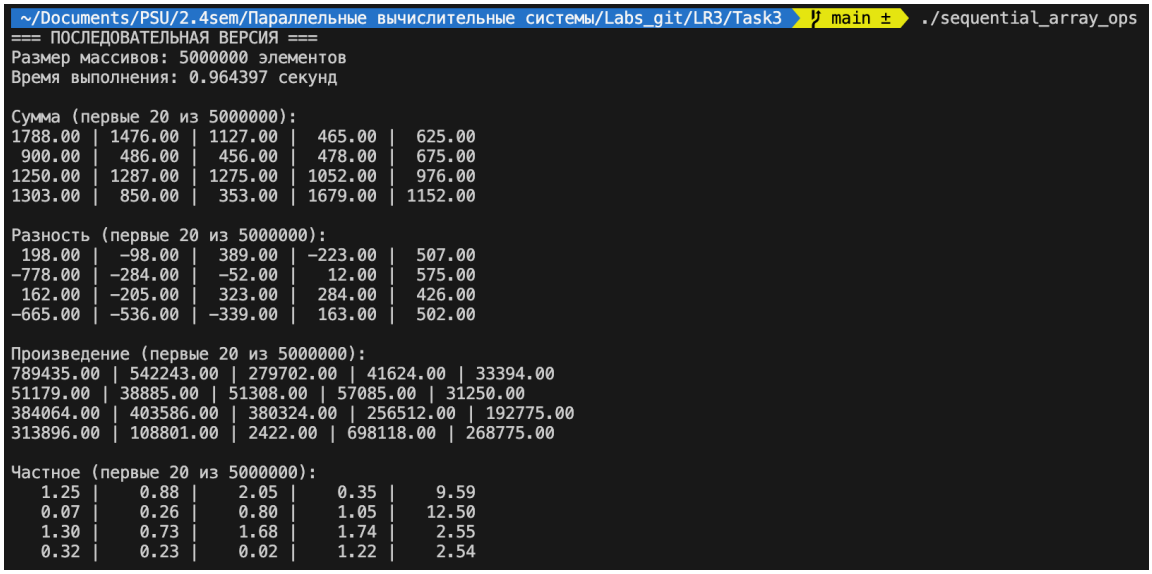




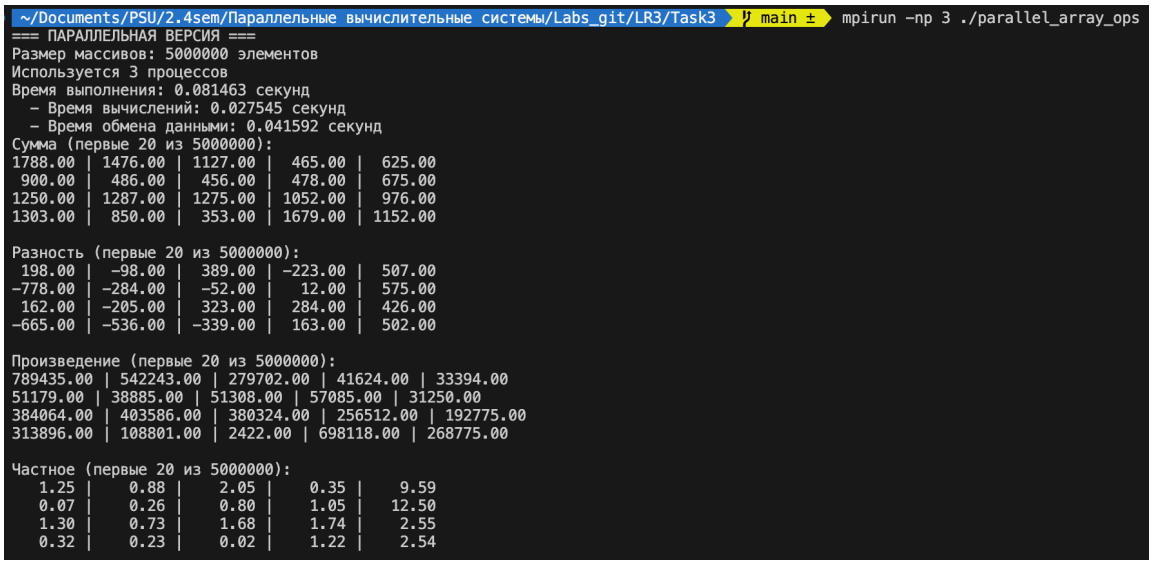
Здесь также видна разница по скорости работы программ. Параллельный вариант на 4 процесса быстрее последовательного в 16+ раз, и при увеличении количества процессов скорость работы программы также планомерно увеличивалась. Но скорость работы параллельной программы, запущенной локально все равно оказалась быстрее. Это демонстрирует эффективность распараллеливания задачи сортировки, хотя с увеличением числа процессов прирост производительности снижается из-за накладных расходов на коммуникации между процессами.

Task 3

- В третьем задании было необходимо реализовать последовательный и параллельный вариант программы работы с двумя одномерными массивами, в которой будет реализованы операции сложения, вычитания, умножения, деления элементов с одинаковыми индексами, что и было успешно реализовано. Для проверки корректности работы реализовал вывод первых 20 результатов, для каждой из операций. А также функцию для показа скорости операций в секунду. Последовательный вариант работы программы уже был реализован во 2ой лабораторной работе, поэтому я его и использовал, но снова увеличил количество элементов до 5.000.000 для лучшей демонстрации работы. Ниже представлены результаты работы на MacBook.



```
~/Documents/PSU/2.4sem/Параллельные вычислительные системы/Labs_git/LR3/Task3  main ±  ./sequential_array_ops
=== ПОСЛЕДОВАТЕЛЬНАЯ ВЕРСИЯ ===
Размер массивов: 5000000 элементов
Время выполнения: 0.964397 секунд

Сумма (первые 20 из 5000000):
1788.00 | 1476.00 | 1127.00 |  465.00 |  625.00
 900.00 |  486.00 |  456.00 |  478.00 |  675.00
1250.00 | 1287.00 | 1275.00 | 1052.00 |  976.00
1303.00 |  850.00 |  353.00 | 1679.00 | 1152.00

Разность (первые 20 из 5000000):
 198.00 |  -98.00 |  389.00 | -223.00 |  507.00
-778.00 | -284.00 |  -52.00 |   12.00 |  575.00
 162.00 | -205.00 |  323.00 |  284.00 |  426.00
-665.00 | -536.00 | -339.00 |  163.00 |  502.00

Произведение (первые 20 из 5000000):
789435.00 | 542243.00 | 279702.00 | 41624.00 | 33394.00
51179.00 | 38885.00 | 51308.00 | 57085.00 | 31250.00
384064.00 | 403586.00 | 380324.00 | 256512.00 | 192775.00
313896.00 | 108801.00 | 2422.00 | 698118.00 | 268775.00

Частное (первые 20 из 5000000):
   1.25 |    0.88 |    2.05 |    0.35 |    9.59
   0.07 |    0.26 |    0.80 |    1.05 |   12.50
   1.30 |    0.73 |    1.68 |    1.74 |    2.55
   0.32 |    0.23 |    0.02 |    1.22 |    2.54
```



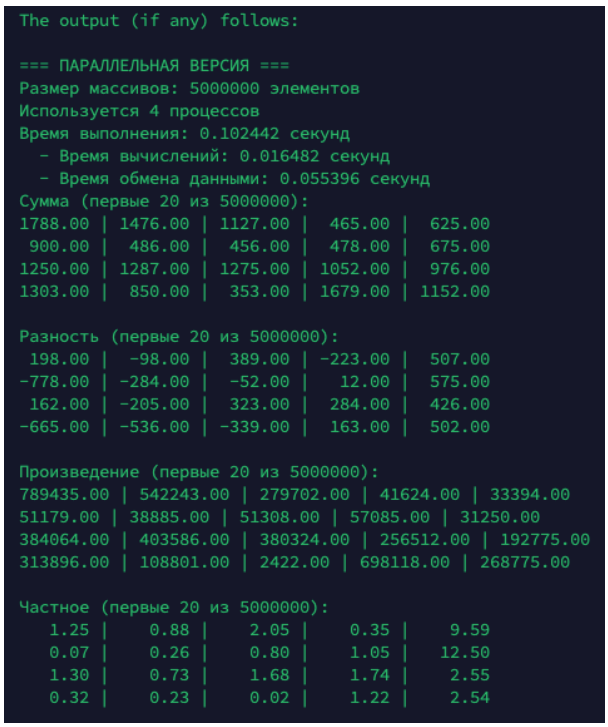
```
~/Documents/PSU/2.4sem/Параллельные вычислительные системы/Labs_git/LR3/Task3  main ±  mpirun -np 3 ./parallel_array_ops
=== ПАРАЛЛЕЛЬНАЯ ВЕРСИЯ ===
Размер массивов: 5000000 элементов
Используется 3 процессов
Время выполнения: 0.081463 секунд
  - Время вычислений: 0.027545 секунд
  - Время обмена данными: 0.041592 секунд
Сумма (первые 20 из 5000000):
1788.00 | 1476.00 | 1127.00 |  465.00 |  625.00
 900.00 |  486.00 |  456.00 |  478.00 |  675.00
1250.00 | 1287.00 | 1275.00 | 1052.00 |  976.00
1303.00 |  850.00 |  353.00 | 1679.00 | 1152.00

Разность (первые 20 из 5000000):
 198.00 |  -98.00 |  389.00 | -223.00 |  507.00
-778.00 | -284.00 |  -52.00 |   12.00 |  575.00
 162.00 | -205.00 |  323.00 |  284.00 |  426.00
-665.00 | -536.00 | -339.00 |  163.00 |  502.00

Произведение (первые 20 из 5000000):
789435.00 | 542243.00 | 279702.00 | 41624.00 | 33394.00
51179.00 | 38885.00 | 51308.00 | 57085.00 | 31250.00
384064.00 | 403586.00 | 380324.00 | 256512.00 | 192775.00
313896.00 | 108801.00 | 2422.00 | 698118.00 | 268775.00

Частное (первые 20 из 5000000):
   1.25 |    0.88 |    2.05 |    0.35 |    9.59
   0.07 |    0.26 |    0.80 |    1.05 |   12.50
   1.30 |    0.73 |    1.68 |    1.74 |    2.55
   0.32 |    0.23 |    0.02 |    1.22 |    2.54
```

```
~/Documents/PSU/2.4sem/Параллельные вычислительные системы/Labs_git/LR3/Task3  main ±  mpirun -np 4 ./parallel_array_ops
=== ПАРАЛЛЕЛЬНАЯ ВЕРСИЯ ===
Размер массивов: 5000000 элементов
Используется 4 процессов
Время выполнения: 0.058258 секунд
  - Время вычислений: 0.006819 секунд
  - Время обмена данными: 0.038608 секунд
Сумма (первые 20 из 5000000):
1788.00 | 1476.00 | 1127.00 |  465.00 |  625.00
 900.00 |  486.00 |  456.00 |  478.00 |  675.00
1250.00 | 1287.00 | 1275.00 | 1052.00 |  976.00
1303.00 |  850.00 |  353.00 | 1679.00 | 1152.00

Разность (первые 20 из 5000000):
 198.00 |  -98.00 |  389.00 | -223.00 |  507.00
-778.00 | -284.00 |  -52.00 |   12.00 |  575.00
 162.00 | -205.00 |  323.00 |  284.00 |  426.00
-665.00 | -536.00 | -339.00 |  163.00 |  502.00

Произведение (первые 20 из 5000000):
789435.00 | 542243.00 | 279702.00 | 41624.00 | 33394.00
51179.00 | 38885.00 | 51308.00 | 57085.00 | 31250.00
384064.00 | 403586.00 | 380324.00 | 256512.00 | 192775.00
313896.00 | 108801.00 | 2422.00 | 698118.00 | 268775.00

Частное (первые 20 из 5000000):
   1.25 |    0.88 |    2.05 |    0.35 |    9.59
   0.07 |    0.26 |    0.80 |    1.05 |   12.50
   1.30 |    0.73 |    1.68 |    1.74 |    2.55
   0.32 |    0.23 |    0.02 |    1.22 |    2.54
```

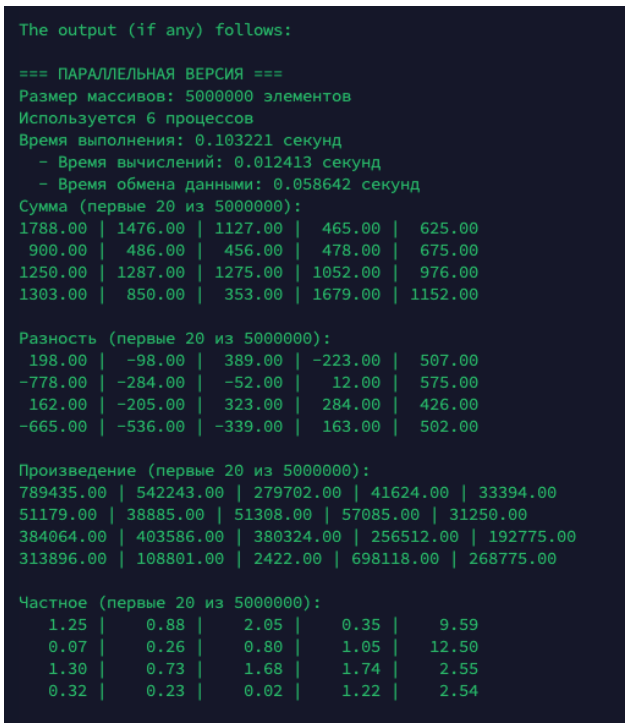
Опять же видим очень хорошие результаты. При использовании 3 процессов ускорение составляет 11.8 раз, по сравнению с последовательным вариантом. А при использовании 6 процессов, скорость работы увеличивается еще на 40%.

- Результат выполнения программ на «ПГНИУ-Кеплер»

```
The output (if any) follows:

=== ПОСЛЕДОВАТЕЛЬНАЯ ВЕРСИЯ ===
Размер массивов: 5000000 элементов
Время выполнения: 1.622853 секунд

Сумма (первые 20 из 5000000):
1788.00 | 1476.00 | 1127.00 |  465.00 |  625.00
 900.00 |  486.00 |  456.00 |  478.00 |  675.00
1250.00 | 1287.00 | 1275.00 | 1052.00 |  976.00
1303.00 |  850.00 |  353.00 | 1679.00 | 1152.00

Разность (первые 20 из 5000000):
 198.00 |  -98.00 |  389.00 | -223.00 |  507.00
-778.00 | -284.00 |  -52.00 |   12.00 |  575.00
 162.00 | -205.00 |  323.00 |  284.00 |  426.00
-665.00 | -536.00 | -339.00 |  163.00 |  502.00

Произведение (первые 20 из 5000000):
789435.00 | 542243.00 | 279702.00 | 41624.00 | 33394.00
51179.00 | 38885.00 | 51308.00 | 57085.00 | 31250.00
384064.00 | 403586.00 | 380324.00 | 256512.00 | 192775.00
313896.00 | 108801.00 | 2422.00 | 698118.00 | 268775.00

Частное (первые 20 из 5000000):
   1.25 |    0.88 |    2.05 |    0.35 |    9.59
   0.07 |    0.26 |    0.80 |    1.05 |   12.50
   1.30 |    0.73 |    1.68 |    1.74 |    2.55
   0.32 |    0.23 |    0.02 |    1.22 |    2.54
```



```
The output (if any) follows:

=== ПАРАЛЛЕЛЬНАЯ ВЕРСИЯ ===
Размер массивов: 5000000 элементов
Используется 4 процессов
Время выполнения: 0.102442 секунд
  - Время вычислений: 0.016482 секунд
  - Время обмена данными: 0.055396 секунд
Сумма (первые 20 из 5000000):
1788.00 | 1476.00 | 1127.00 |  465.00 |  625.00
 900.00 |  486.00 |  456.00 |  478.00 |  675.00
1250.00 | 1287.00 | 1275.00 | 1052.00 |  976.00
1303.00 |  850.00 |  353.00 | 1679.00 | 1152.00

Разность (первые 20 из 5000000):
 198.00 |  -98.00 |  389.00 | -223.00 |  507.00
-778.00 | -284.00 |  -52.00 |   12.00 |  575.00
 162.00 | -205.00 |  323.00 |  284.00 |  426.00
-665.00 | -536.00 | -339.00 |  163.00 |  502.00

Произведение (первые 20 из 5000000):
789435.00 | 542243.00 | 279702.00 | 41624.00 | 33394.00
51179.00 | 38885.00 | 51308.00 | 57085.00 | 31250.00
384064.00 | 403586.00 | 380324.00 | 256512.00 | 192775.00
313896.00 | 108801.00 | 2422.00 | 698118.00 | 268775.00

Частное (первые 20 из 5000000):
   1.25 |    0.88 |    2.05 |    0.35 |    9.59
   0.07 |    0.26 |    0.80 |    1.05 |   12.50
   1.30 |    0.73 |    1.68 |    1.74 |    2.55
   0.32 |    0.23 |    0.02 |    1.22 |    2.54
```



```
The output (if any) follows:

=== ПАРАЛЛЕЛЬНАЯ ВЕРСИЯ ===
Размер массивов: 5000000 элементов
Используется 6 процессов
Время выполнения: 0.103221 секунд
  - Время вычислений: 0.012413 секунд
  - Время обмена данными: 0.058642 секунд
Сумма (первые 20 из 5000000):
1788.00 | 1476.00 | 1127.00 |  465.00 |  625.00
 900.00 |  486.00 |  456.00 |  478.00 |  675.00
1250.00 | 1287.00 | 1275.00 | 1052.00 |  976.00
1303.00 |  850.00 |  353.00 | 1679.00 | 1152.00

Разность (первые 20 из 5000000):
 198.00 |  -98.00 |  389.00 | -223.00 |  507.00
-778.00 | -284.00 |  -52.00 |   12.00 |  575.00
 162.00 | -205.00 |  323.00 |  284.00 |  426.00
-665.00 | -536.00 | -339.00 |  163.00 |  502.00

Произведение (первые 20 из 5000000):
789435.00 | 542243.00 | 279702.00 | 41624.00 | 33394.00
51179.00 | 38885.00 | 51308.00 | 57085.00 | 31250.00
384064.00 | 403586.00 | 380324.00 | 256512.00 | 192775.00
313896.00 | 108801.00 | 2422.00 | 698118.00 | 268775.00

Частное (первые 20 из 5000000):
   1.25 |    0.88 |    2.05 |    0.35 |    9.59
   0.07 |    0.26 |    0.80 |    1.05 |   12.50
   1.30 |    0.73 |    1.68 |    1.74 |    2.55
   0.32 |    0.23 |    0.02 |    1.22 |    2.54
```

Также видим ожидаемые результаты. Последовательный вариант оказался медленнее всего. А вариант на 4 и 6 процессов кратно быстрее. Но быстрее всего опять же оказался параллельный вариант, запущенный локально.

Task 4

- В 4ом задании было необходимо написать последовательный и параллельный вариант программы работы с двумя двумерными массивами: операции сложения, вычитания, умножения, деления элементов с одинаковыми индексами. Здесь я также увеличил количество тестируемых элементов, в сравнении с предыдущей лабораторной работой до 4 миллионов. Для этого немного изменил скрипт на python.

```python
import random
import math

rows, cols = 2000, 2000

# Создаем двумерный массив
matrix = [[random.randint(1, 1000) for _ in range(cols)] for _ in range(rows)]

# Сохраняем матрицу в файл
with open("matrix2.txt", "w") as f:
    # Сначала записываем размеры матрицы
    f.write(f"{rows} {cols}\n")

```

```
      # Затем записываем саму матрицу
      for row in matrix:
          f.write(" ".join(map(str, row)) + "\n")

  print(f"Матрица {rows}x{cols} (всего {rows*cols} элементов) сохранена в файл
  matrix2.txt")

```

- Ниже представлены результаты работы на MacBook.

```
~/Documents/PSU/2.4sem/Параллельные вычислительные системы/Labs_git/LR3/Task4 main ± ./sequential_matrix_ops
=== ПОСЛЕДОВАТЕЛЬНАЯ ВЕРСИЯ ===
Размер матриц: 2000x2000 (всего 4000000 элементов)
Время выполнения: 0.482042 секунд
Скорость: 33192128.49 операций/сек
Первые 5 результатов операций:
Индекс |   Сложение   |  Вычитание   | Умножение   |  Деление
-------+-------------+-------------+-------------+-------------
     1 |      702.00 |      568.00 |   42545.00 |        9.48
     2 |      684.00 |     -638.00 |   15203.00 |        0.03
     3 |      863.00 |     -653.00 |   79590.00 |        0.14
     4 |      286.00 |     -180.00 |   12349.00 |        0.23
     5 |     1313.00 |      657.00 |  323080.00 |        3.00

Всего обработано элементов: 4000000

~/Documents/PSU/2.4sem/Параллельные вычислительные системы/Labs_git/LR3/Task4 main ± mpirun -np 2 ./parallel_matrix_ops
=== ПАРАЛЛЕЛЬНАЯ ВЕРСИЯ ===
Размер матриц: 2000x2000 (всего 4000000 элементов)
Используется 2 процессов
Время выполнения: 0.064201 секунд
  - Время вычислений: 0.010614 секунд
  - Время обмена данными: 0.035609 секунд
Скорость: 249217301.91 операций/сек
Первые 5 результатов операций:
Индекс |   Сложение   |  Вычитание   | Умножение   |  Деление
-------+-------------+-------------+-------------+-------------
     1 |      702.00 |      568.00 |   42545.00 |        9.48
     2 |      684.00 |     -638.00 |   15203.00 |        0.03
     3 |      863.00 |     -653.00 |   79590.00 |        0.14
     4 |      286.00 |     -180.00 |   12349.00 |        0.23
     5 |     1313.00 |      657.00 |  323080.00 |        3.00

Всего обработано элементов: 4000000
```

Как видим, параллельный вариант всего лишь на 2 процесса оказался в разы быстрее классического последовательного варианта.

- Результат выполнения работы программы на «ПГНИУ-Кеплер»

```
The output (if any) follows:

=== ПОСЛЕДОВАТЕЛЬНАЯ ВЕРСИЯ ===
Размер матриц: 2000x2000 (всего 4000000 элементов)
Время выполнения: 1.101792 секунд
Скорость: 14521797.22 операций/сек
Первые 5 результатов операций:
Индекс |   Сложение   |  Вычитание   | Умножение   |  Деление
-------+-------------+-------------+-------------+-------------
     1 |      702.00 |      568.00 |   42545.00 |        9.48
     2 |      684.00 |     -638.00 |   15203.00 |        0.03
     3 |      863.00 |     -653.00 |   79590.00 |        0.14
     4 |      286.00 |     -180.00 |   12349.00 |        0.23
     5 |     1313.00 |      657.00 |  323080.00 |        3.00

Всего обработано элементов: 4000000
```

```
The output (if any) follows:

=== ПАРАЛЛЕЛЬНАЯ ВЕРСИЯ ===
Размер матриц: 2000x2000 (всего 4000000 элементов)
Используется 4 процессов
Время выполнения: 0.155998 секунд
  - Время вычислений: 0.016577 секунд
  - Время обмена данными: 0.085743 секунд
Скорость: 102565391.86 операций/сек
Первые 5 результатов операций:
Индекс |   Сложение   |  Вычитание   | Умножение   |  Деление
-------+--------------+--------------+-------------+-------------
     1 |      702.00 |      568.00 |   42545.00 |       9.48
     2 |      684.00 |     -638.00 |   15203.00 |       0.03
     3 |      863.00 |     -653.00 |   79590.00 |       0.14
     4 |      286.00 |     -180.00 |   12349.00 |       0.23
     5 |     1313.00 |      657.00 |  323080.00 |       3.00

Всего обработано элементов: 4000000
```

```
The output (if any) follows:

=== ПАРАЛЛЕЛЬНАЯ ВЕРСИЯ ===
Размер матриц: 2000x2000 (всего 4000000 элементов)
Используется 10 процессов
Время выполнения: 0.149666 секунд
  - Время вычислений: 0.006988 секунд
  - Время обмена данными: 0.092920 секунд
Скорость: 106904682.10 операций/сек
Первые 5 результатов операций:
Индекс |   Сложение   |  Вычитание   | Умножение   |  Деление
-------+--------------+--------------+-------------+-------------
     1 |      702.00 |      568.00 |   42545.00 |       9.48
     2 |      684.00 |     -638.00 |   15203.00 |       0.03
     3 |      863.00 |     -653.00 |   79590.00 |       0.14
     4 |      286.00 |     -180.00 |   12349.00 |       0.23
     5 |     1313.00 |      657.00 |  323080.00 |       3.00

Всего обработано элементов: 4000000
```

Параллельная версия демонстрирует отличную масштабируемость, превзойдя линейное ускорение (7.35x при 10 процессах). Это может быть связано с оптимизациями компилятора и эффективным использованием кеша при разделении данных между процессами. Результаты операций идентичны, что подтверждает корректность работы параллельной реализации.

Вывод

В ходе выполнения лабораторной работы были изучены основы параллельного программирования с использованием технологии MPI (Message Passing Interface). Основные результаты:

1. **Реализованы параллельные версии** для следующих задач:
    - Суммирование элементов массива
    - Сортировка массива (сортировка пузырьком)
    - Операции с матрицами (сложение, вычитание, умножение, деление)
2. **Проведено сравнение** производительности последовательных и параллельных версий программ:
    - Продемонстрирована эффективность MPI при обработке больших объемов данных
    - Выявлены накладные расходы на передачу сообщений между процессами
    - Определены оптимальные размеры задач для эффективного распараллеливания
3. **Особенности реализации**:
    - Использованы директивы OpenMP для распараллеливания циклов
    - Реализована проверка корректности работы параллельных алгоритмов
    - Добавлен замер времени выполнения для оценки эффективности
4. **Вывод по эффективности**:
    - MPI демонстрирует высокую эффективность при обработке больших массивов данных
    - Наибольший прирост производительности наблюдается при работе с задачами, где объем вычислений значительно превышает объем передаваемых данных
    - Оптимальное количество процессов зависит от размера задачи и характеристик вычислительной системы
5. **Преимущества подхода**:

- Возможность масштабирования на вычислительные кластеры
- Эффективная работа с данными, не помещающимися в память одного узла
- Гибкость в распределении вычислительной нагрузки

Работа позволила получить практические навыки разработки распределенных приложений с использованием MPI, а также провести анализ эффективности параллельных алгоритмов в зависимости от размера задачи и количества задействованных процессов.

Теоретическая справка

1. Введение в технологию MPI

1.1 Основные понятия

MPI (Message Passing Interface) - это стандартизированный программный интерфейс для организации межпроцессного взаимодействия в параллельных вычислениях. В отличие от OpenMP, который использует общую память, MPI основан на передаче сообщений между независимыми процессами.

1.2 Архитектура MPI

- **Процессы** - независимые экземпляры программы, выполняющиеся параллельно
- **Коммуникатор** (MPI_Comm) - группа процессов, которые могут взаимодействовать
- **Ранги** - уникальные идентификаторы процессов в коммуникаторе
- **Теги** - метки для различения типов сообщений

OpenMP

- **Параллелизм на уровне потоков** (нитей)
- Работает в рамках одного процесса с общей памятью
- Потоки делят общие данные

- Простая реализация с помощью директив компилятора (#pragma omp)
- Идеален для многоядерных систем с общей памятью

MPI (Message Passing Interface)

- **Параллелизм на уровне процессов**
- Каждый процесс имеет свою собственную память
- Обмен данными через передачу сообщений
- Требует явного программирования обмена данными
- Масштабируется на кластеры с распределенной памятью

Ключевые отличия:

- **Память**:

OpenMP: общая память (shared memory)

MPI: распределенная память (distributed memory)

- **Масштабируемость**:

OpenMP: ограничено одним узлом (одной машиной)

MPI: может работать на тысячах узлов

- **Сложность программирования**:

OpenMP: проще, меньше кода

MPI: сложнее, больше кода для обмена данными

- **Применение**:

OpenMP: подходит для циклов и участков кода с общей памятью

MPI: для задач, требующих работы с большими объемами данных на разных узлах

MPI:

1. Запускается несколько независимых копий программы (процессов)
2. Каждый процесс работает независимо и имеет свою собственную память
3. Процессы могут выполняться как на одном компьютере, так и на разных
4. Количество процессов задается при запуске программы
5. Каждый процесс может выполняться на отдельном ядре или даже на отдельном компьютере