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 № 2

Название: Введение в OpenMP. Операции с массивами. Быстрая 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 OpenMP.

Задание работы

1. Изучить основы технологии параллельного программирования средствами библиотеки OpenMP.
2. Изучить способы разделения задач между потоками.
3. Способом параллельных циклов научиться решать задачи сложения (и других арифметических операций) элементов.
4. Способом параллельных задач научиться реализовывать параллельную работу рекурсивных функций на примере быстрой сортировки элементов.

Основная часть

- Первым делом необходимо сгенерировать массив, в котором количество элементов будет больше 100.000. Я сделал это при помощи python скрипта, который генерирует массив из 100.001 элемента.

```python
import random

# Генерируем массив из 100001 случайного числа от 1 до 100
arr = [random.randint(1, 100) for _ in range(100001)]

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

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

- Далее были созданы последовательный и параллельный вариант выполнения указанной программы на языке C. Также для теста написал последовательный вариант на python. Весь дальнейший код программ представлен на Github в моем репозитории.

Task 1

- Результат выполнения программ в локальной среде программирования

```
~/Documents/PSU/2.4sem/Параллельные вычислительные системы/Labs_git/LR2/Task1  main ± python Task1_sequential.py
Сумма элементов = 5037087
Среднее время выполнения (5 запусков): 0.001661 сек
~/Documents/PSU/2.4sem/Параллельные вычислительные системы/Labs_git/LR2/Task1  main ± ./sequential_sum
Размер массива: 100001 элементов
Сумма элементов: 5037087
Время выполнения: 0.012719 секунд
~/Documents/PSU/2.4sem/Параллельные вычислительные системы/Labs_git/LR2/Task1  main ± ./parallel_sum 4
Количество потоков: 4
Размер массива: 100001 элементов
Сумма элементов: 5037087
Время выполнения: 0.014730 секунд
~/Documents/PSU/2.4sem/Параллельные вычислительные системы/Labs_git/LR2/Task1  main ± ./parallel_sum 8
Количество потоков: 8
Размер массива: 100001 элементов
Сумма элементов: 5037087
Время выполнения: 0.016561 секунд
```

Здесь мы видим, что для массивов такого размера (100,001 элементов) накладные расходы на создание потоков превышают выгоду от параллелизма. Python показал лучшую производительность благодаря внутренним оптимизациям.

- Результат выполнения программ на «ПГНИУ-Кеплер»

```
Your job looked like:

------------------------------------------------------------
# LSBATCH: User input
#!/bin/bash
#BSUB -J ParSum
#BSUB -P ParallelComputing
#BSUB -W 00:01
#BSUB -n 2
#BSUB -oo par_output.log
#BSUB -eo par_error.log

export OMP_NUM_THREADS 8
./parallel_sum 8

------------------------------------------------------------

Successfully completed.

Resource usage summary:

    CPU time :                                   0.05 sec.
    Max Memory :                                 -
    Average Memory :                             -
    Total Requested Memory :                     -
    Delta Memory :                               -
    Max Swap :                                   -
    Max Processes :                              -
    Max Threads :                                -
    Run time :                                   7 sec.
    Turnaround time :                            3 sec.

The output (if any) follows:

Количество потоков: 8
Размер массива: 100001 элементов
Сумма элементов: 5037087
Время выполнения: 0.017034 секунд
```

```
Your job looked like:

------------------------------------------------------------
# LSBATCH: User input
#!/bin/bash
#BSUB -J ParSum
#BSUB -P ParallelComputing
#BSUB -W 00:01
#BSUB -n 2
#BSUB -oo par_output.log
#BSUB -eo par_error.log

export OMP_NUM_THREADS=4
./parallel_sum 4

------------------------------------------------------------

Successfully completed.

Resource usage summary:

    CPU time :                                   0.05 sec.
    Max Memory :                                 -
    Average Memory :                             -
    Total Requested Memory :                     -
    Delta Memory :                               -
    Max Swap :                                   -
    Max Processes :                              -
    Max Threads :                                -
    Run time :                                   7 sec.
    Turnaround time :                            5 sec.

The output (if any) follows:

Количество потоков: 4
Размер массива: 100001 элементов
Сумма элементов: 5037087
Время выполнения: 0.017084 секунд
```

```
Successfully completed.

Resource usage summary:

    CPU time :                                   0.10 sec.
    Max Memory :                                 6 MB
    Average Memory :                             4.33 MB
    Total Requested Memory :                     -
    Delta Memory :                               -
    Max Swap :                                   -
    Max Processes :                              1
    Max Threads :                                1
    Run time :                                   13 sec.
    Turnaround time :                            21 sec.

The output (if any) follows:

Размер массива: 100001 элементов
Сумма элементов: 5037087
Время выполнения: 0.015489 секунд
```

Последовательный вариант оказался даже чуть быстрее параллельного. Параллельная версия с 8 потоками завершила работу чуть быстрее, чем на 4ех потоках. Быстрее всего в этом задании оказался последовательный вариант, запущенный в своей среде разработки.

Task 2

- В заданиях №1, №2 и №3 используем массив, созданный в самом начале, чтобы увеличить точность сравнения работы различных вариантов программ. Ниже представлены результаты выполнения программ быстрой сортировки массива на локальном компьютере.

```
~/Documents/PSU/2.4sem/Параллельные вычислительные системы/Labs_git/LR2/Task2  main ± ./sequential_quicksort
Последовательная быстрая сортировка
Размер массива: 100001 элементов
Время выполнения: 0.047408 секунд
```

```
~/Documents/PSU/2.4sem/Параллельные вычислительные системы/Labs_git/LR2/Task2  main ± ./parallel_quicksort 2 1000
Параллельная быстрая сортировка
Количество потоков: 2
Порог параллелизма: 1000
Размер массива: 100001 элементов
Время выполнения: 0.049715 секунд
~/Documents/PSU/2.4sem/Параллельные вычислительные системы/Labs_git/LR2/Task2  main ± ./parallel_quicksort 6 1000
Параллельная быстрая сортировка
Количество потоков: 6
Порог параллелизма: 1000
Размер массива: 100001 элементов
Время выполнения: 0.029344 секунд
~/Documents/PSU/2.4sem/Параллельные вычислительные системы/Labs_git/LR2/Task2  main ± ./parallel_quicksort 10 1000
Параллельная быстрая сортировка
Количество потоков: 10
Порог параллелизма: 1000
Размер массива: 100001 элементов
Время выполнения: 0.024600 секунд
```

Здесь уже видим более логичные результаты по скорости работы программы. Параллельный (2 потока) > последовательный > параллельный (6 потоков) > параллельный (10 потоков).

- Результат выполнения программ на «ПГНИУ-Кеплер»

```
gcc -fopenmp -o parallel_quicksort parallel_quicksort.c -lm
export OMP_NUM_THREADS=4
./parallel_quicksort 4 1000

------------------------------------------------------------

Successfully completed.

Resource usage summary:

    CPU time :                                   0.50 sec.
    Max Memory :                                 8 MB
    Average Memory :                             5.33 MB
    Total Requested Memory :                     -
    Delta Memory :                               -
    Max Swap :                                   -
    Max Processes :                              5
    Max Threads :                                6
    Run time :                                   3 sec.
    Turnaround time :                            2 sec.

The output (if any) follows:

Параллельная быстрая сортировка
Количество потоков: 4
Порог параллелизма: 1000
Размер массива: 100001 элементов
Время выполнения: 0.222371 секунд
```

```
gcc -fopenmp -o parallel_quicksort parallel_quicksort.c -lm
export OMP_NUM_THREADS=8
./parallel_quicksort 8 1000

------------------------------------------------------------

Successfully completed.

Resource usage summary:

    CPU time :                                   0.55 sec.
    Max Memory :                                 10 MB
    Average Memory :                             7.00 MB
    Total Requested Memory :                     -
    Delta Memory :                               -
    Max Swap :                                   -
    Max Processes :                              5
    Max Threads :                                6
    Run time :                                   8 sec.
    Turnaround time :                            6 sec.

The output (if any) follows:

Параллельная быстрая сортировка
Количество потоков: 8
Порог параллелизма: 1000
Размер массива: 100001 элементов
Время выполнения: 0.128171 секунд
```

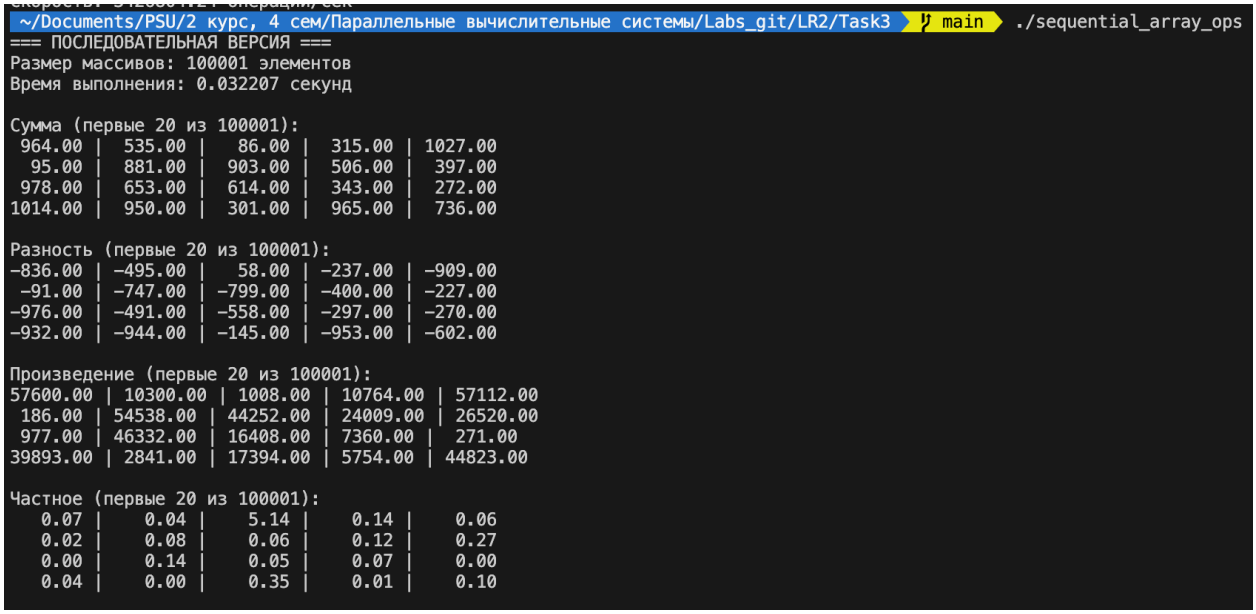
```
./sequential_quicksort

------------------------------------------------------------

Successfully completed.

Resource usage summary:

    CPU time :                                   0.43 sec.
    Max Memory :                                 -
    Average Memory :                             -
    Total Requested Memory :                     -
    Delta Memory :                               -
    Max Swap :                                   -
    Max Processes :                              -
    Max Threads :                                -
    Run time :                                   3 sec.
    Turnaround time :                            5 sec.

The output (if any) follows:

Последовательная быстрая сортировка
Размер массива: 100001 элементов
Время выполнения: 0.402286 секунд
```

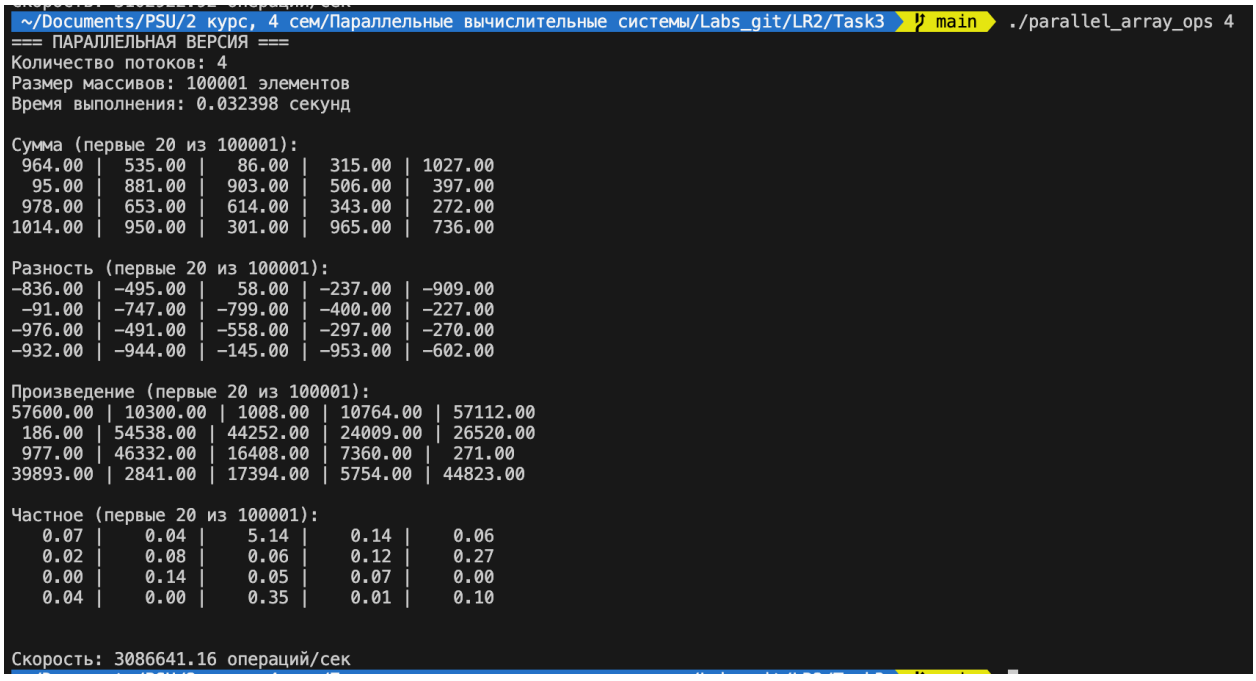
Здесь все ожидаемо: параллельный вариант (8 потоков) < параллельный вариант (4 потока) < последовательный вариант.

По итогу самым эффективным оказался параллельный вариант на 8 потоков, запущенный на «ПГНИУ-Кеплер».

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. Ниже представлены результаты работы на MacBook.





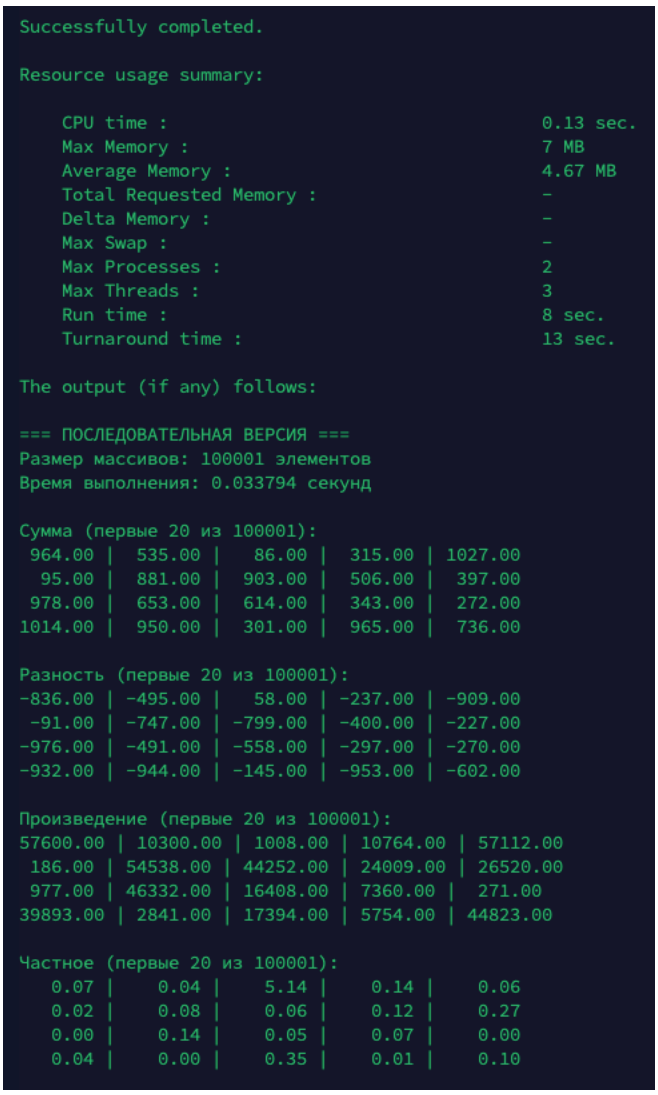
```
~/Documents/PSU/2 курс, 4 сем/Параллельные вычислительные системы/Labs_git/LR2/Task3  main  ./parallel_array_ops 8
=== ПАРАЛЛЕЛЬНАЯ ВЕРСИЯ ===
Количество потоков: 8
Размер массивов: 100001 элементов
Время выполнения: 0.026652 секунд

Сумма (первые 20 из 100001):
 964.00 |  535.00 |   86.00 |  315.00 | 1027.00
  95.00 |  881.00 |  903.00 |  506.00 |  397.00
 978.00 |  653.00 |  614.00 |  343.00 |  272.00
1014.00 |  950.00 |  301.00 |  965.00 |  736.00

Разность (первые 20 из 100001):
-836.00 | -495.00 |   58.00 | -237.00 | -909.00
 -91.00 | -747.00 | -799.00 | -400.00 | -227.00
-976.00 | -491.00 | -558.00 | -297.00 | -270.00
-932.00 | -944.00 | -145.00 | -953.00 | -602.00

Произведение (первые 20 из 100001):
57600.00 | 10300.00 | 1008.00 | 10764.00 | 57112.00
 186.00 | 54538.00 | 44252.00 | 24009.00 | 26520.00
 977.00 | 46332.00 | 16408.00 | 7360.00 |  271.00
39893.00 | 2841.00 | 17394.00 | 5754.00 | 44823.00

Частное (первые 20 из 100001):
   0.07 |    0.04 |    5.14 |    0.14 |    0.06
   0.02 |    0.08 |    0.06 |    0.12 |    0.27
   0.00 |    0.14 |    0.05 |    0.07 |    0.00
   0.04 |    0.00 |    0.35 |    0.01 |    0.10


Скорость: 3752101.17 операций/сек
```

Параллельный вариант (4 потока) > последовательный > параллельный (8 потоков).

- Результат выполнения программ на «ПГНИУ-Кеплер»

```
Successfully completed.

Resource usage summary:

    CPU time :                                   0.14 sec.
    Max Memory :                                 8 MB
    Average Memory :                             8.00 MB
    Total Requested Memory :                     -
    Delta Memory :                               -
    Max Swap :                                   -
    Max Processes :                              5
    Max Threads :                                6
    Run time :                                   8 sec.
    Turnaround time :                            4 sec.

The output (if any) follows:

=== ПАРАЛЛЕЛЬНАЯ ВЕРСИЯ ===
Количество потоков: 4
Размер массивов: 100001 элементов
Время выполнения: 0.051220 секунд

Сумма (первые 20 из 100001):
 964.00 |  535.00 |   86.00 |  315.00 | 1027.00
  95.00 |  881.00 |  903.00 |  506.00 |  397.00
 978.00 |  653.00 |  614.00 |  343.00 |  272.00
1014.00 |  950.00 |  301.00 |  965.00 |  736.00

Разность (первые 20 из 100001):
-836.00 | -495.00 |   58.00 | -237.00 | -909.00
 -91.00 | -747.00 | -799.00 | -400.00 | -227.00
-976.00 | -491.00 | -558.00 | -297.00 | -270.00
-932.00 | -944.00 | -145.00 | -953.00 | -602.00

Произведение (первые 20 из 100001):
57600.00 | 10300.00 | 1008.00 | 10764.00 | 57112.00
 186.00 | 54538.00 | 44252.00 | 24009.00 | 26520.00
 977.00 | 46332.00 | 16408.00 | 7360.00 |  271.00
39893.00 | 2841.00 | 17394.00 | 5754.00 | 44823.00

Частное (первые 20 из 100001):
   0.07 |    0.04 |    5.14 |    0.14 |    0.06
   0.02 |    0.08 |    0.06 |    0.12 |    0.27
   0.00 |    0.14 |    0.05 |    0.07 |    0.00
   0.04 |    0.00 |    0.35 |    0.01 |    0.10


Скорость: 1952383.31 операций/сек
```

```
Successfully completed.

Resource usage summary:

    CPU time :                                   0.15 sec.
    Max Memory :                                 10 MB
    Average Memory :                             10.00 MB
    Total Requested Memory :                     -
    Delta Memory :                               -
    Max Swap :                                   -
    Max Processes :                              6
    Max Threads :                                7
    Run time :                                   8 sec.
    Turnaround time :                            4 sec.

The output (if any) follows:

=== ПАРАЛЛЕЛЬНАЯ ВЕРСИЯ ===
Количество потоков: 8
Размер массивов: 100001 элементов
Время выполнения: 0.043198 секунд

Сумма (первые 20 из 100001):
 964.00 |  535.00 |   86.00 |  315.00 | 1027.00
  95.00 |  881.00 |  903.00 |  506.00 |  397.00
 978.00 |  653.00 |  614.00 |  343.00 |  272.00
1014.00 |  950.00 |  301.00 |  965.00 |  736.00

Разность (первые 20 из 100001):
-836.00 | -495.00 |   58.00 | -237.00 | -909.00
 -91.00 | -747.00 | -799.00 | -400.00 | -227.00
-976.00 | -491.00 | -558.00 | -297.00 | -270.00
-932.00 | -944.00 | -145.00 | -953.00 | -602.00

Произведение (первые 20 из 100001):
57600.00 | 10300.00 | 1008.00 | 10764.00 | 57112.00
 186.00 | 54538.00 | 44252.00 | 24009.00 | 26520.00
 977.00 | 46332.00 | 16408.00 | 7360.00 |  271.00
39893.00 | 2841.00 | 17394.00 | 5754.00 | 44823.00

Частное (первые 20 из 100001):
   0.07 |    0.04 |    5.14 |    0.14 |    0.06
   0.02 |    0.08 |    0.06 |    0.12 |    0.27
   0.00 |    0.14 |    0.05 |    0.07 |    0.00
   0.04 |    0.00 |    0.35 |    0.01 |    0.10


Скорость: 2314960.67 операций/сек
```



Последовательный вариант < параллельный (4 потока) < параллельный (8 потоков). Быстрее всего в этом задании оказался последовательный вариант, запущенный в своей среде разработки.

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. Первым делом я написал скрипт на питоне для создания двух матриц размерностью 350 x 350 из 122.500 элементов.

```
import random
import math

rows, cols = 350, 350
```

```python

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

# Сохраняем матрицу в файл
with open("matrix.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 matrix.txt")
```

- Также была написана функция, показывающая скорость выполнения операций в секунду. И вывод первых пяти результатов, для каждой из операция для проверки точности работы программы. Программа проводит вычисления с первыми 50к элементов. Ниже представлены результаты работы на MacBook.

```
~/Documents/PSU/2 курс, 4 сем/Параллельные вычислительные системы/Labs_git/LR2/Task4  main  ./sequential
=== ПОСЛЕДОВАТЕЛЬНАЯ ВЕРСИЯ ===
Размер матриц: 350x350 (всего 122500 элементов)
Время выполнения: 0.032853 секунд
Скорость: 14914924.06 операций/сек
Первые 5 результатов операций:
Индекс |   Сложение   |  Вычитание   | Умножение   |  Деление
-------+--------------+--------------+-------------+-------------
     1 |      1577.00 |       209.00 |  610812.00 |        1.31
     2 |       465.00 |       327.00 |   27324.00 |        5.74
     3 |      1138.00 |      -132.00 |  319405.00 |        0.79
     4 |      1020.00 |       102.00 |  257499.00 |        1.22
     5 |      1047.00 |      -439.00 |  225872.00 |        0.41

Всего обработано элементов: 50000
```

```
~/Documents/PSU/2 курс, 4 сем/Параллельные вычислительные системы/Labs_git/LR2/Task4  main  ./parallel 8
=== ПАРАЛЛЕЛЬНАЯ ВЕРСИЯ (8 потоков) ===
Размер матриц: 350x350 (всего 122500 элементов)
Время выполнения: 0.027860 секунд
Первые 5 результатов операций:
Индекс |   Сложение   |  Вычитание   | Умножение   |  Деление
-------+--------------+--------------+-------------+-------------
     1 |      1577.00 |       209.00 |  610812.00 |        1.31 (поток 0)
     2 |       465.00 |       327.00 |   27324.00 |        5.74 (поток 0)
     3 |      1138.00 |      -132.00 |  319405.00 |        0.79 (поток 0)
     4 |      1483.00 |       205.00 |  539316.00 |        1.32 (поток 3)
     5 |      1020.00 |       102.00 |  257499.00 |        1.22 (поток 0)

Всего обработано элементов: 50000
```

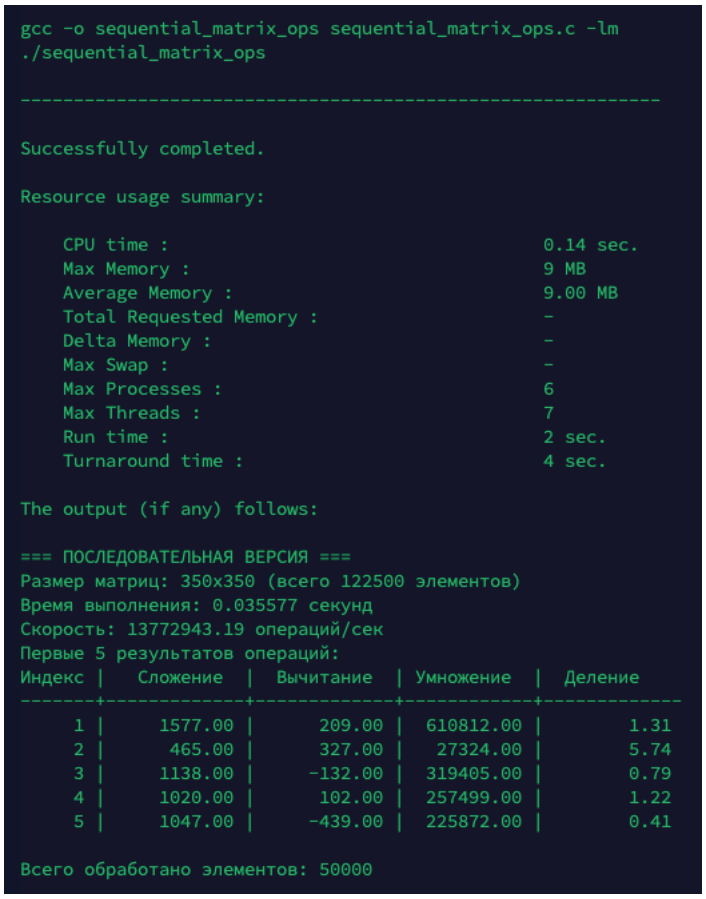
Как видим, параллельный вариант на 8 потоков оказался более выигрышным.

- Результат выполнения параллельного варианта на «ПГНИУ-Кеплер»

```
gcc -fopenmp -o parallel_matrix_ops parallel_matrix_ops.c -lm
export OMP_NUM_THREADS=4
./parallel_matrix_ops 4

------------------------------------------------------------

Successfully completed.

Resource usage summary:

    CPU time :                                   0.16 sec.
    Max Memory :                                 11 MB
    Average Memory :                             11.00 MB
    Total Requested Memory :                     -
    Delta Memory :                               -
    Max Swap :                                   -
    Max Processes :                              6
    Max Threads :                                7
    Run time :                                   8 sec.
    Turnaround time :                            4 sec.

The output (if any) follows:

=== ПАРАЛЛЕЛЬНАЯ ВЕРСИЯ (4 потоков) ===
Размер матриц: 350x350 (всего 122500 элементов)
Время выполнения: 0.045327 секунд
Первые 5 результатов операций:
Индекс |   Сложение   |  Вычитание   | Умножение  |  Деление
-------+--------------+--------------+------------+-------------
     1 |      546.00 |      104.00 |   71825.00 |        1.47 (поток 1)
     2 |      917.00 |     -579.00 |  126412.00 |        0.23 (поток 1)
     3 |      881.00 |      517.00 |  127218.00 |        3.84 (поток 1)
     4 |      851.00 |      643.00 |   77688.00 |        7.18 (поток 1)
     5 |      669.00 |     -191.00 |  102770.00 |        0.56 (поток 1)

Всего обработано элементов: 50000


Скорость: 2702570.10 операций/сек
```

```
gcc -fopenmp -o parallel_matrix_ops parallel_matrix_ops.c -lm
export OMP_NUM_THREADS=8
./parallel_matrix_ops 8

------------------------------------------------------------

Successfully completed.

Resource usage summary:

    CPU time :                                   0.18 sec.
    Max Memory :                                 13 MB
    Average Memory :                             13.00 MB
    Total Requested Memory :                     -
    Delta Memory :                               -
    Max Swap :                                   -
    Max Processes :                              6
    Max Threads :                                7
    Run time :                                   2 sec.
    Turnaround time :                            6 sec.

The output (if any) follows:

=== ПАРАЛЛЕЛЬНАЯ ВЕРСИЯ (8 потоков) ===
Размер матриц: 350x350 (всего 122500 элементов)
Время выполнения: 0.035653 секунд
Первые 5 результатов операций:
Индекс |   Сложение   |  Вычитание   | Умножение  |  Деление
-------+--------------+--------------+------------+-------------
     1 |      879.00 |      205.00 |  182654.00 |        1.61 (поток 1)
     2 |     1483.00 |      205.00 |  539316.00 |        1.32 (поток 3)
     3 |      917.00 |     -579.00 |  126412.00 |        0.23 (поток 2)
     4 |     1577.00 |      209.00 |  610812.00 |        1.31 (поток 0)
     5 |      465.00 |      327.00 |   27324.00 |        5.74 (поток 0)

Всего обработано элементов: 50000


Скорость: 3435901.49 операций/сек
```

- Результат выполнения последовательного варианта на «ПГНИУ-Кеплер»



```
gcc -o sequential_matrix_ops sequential_matrix_ops.c -lm
./sequential_matrix_ops

------------------------------------------------------------

Successfully completed.

Resource usage summary:

    CPU time :                                   0.14 sec.
    Max Memory :                                 9 MB
    Average Memory :                             9.00 MB
    Total Requested Memory :                     -
    Delta Memory :                               -
    Max Swap :                                   -
    Max Processes :                              6
    Max Threads :                                7
    Run time :                                   2 sec.
    Turnaround time :                            4 sec.

The output (if any) follows:

=== ПОСЛЕДОВАТЕЛЬНАЯ ВЕРСИЯ ===
Размер матриц: 350x350 (всего 122500 элементов)
Время выполнения: 0.035577 секунд
Скорость: 13772943.19 операций/сек
Первые 5 результатов операций:
Индекс |   Сложение   |  Вычитание   | Умножение   |  Деление
-------+-------------+-------------+-------------+-------------
     1 |     1577.00 |      209.00 |  610812.00 |        1.31
     2 |      465.00 |      327.00 |   27324.00 |        5.74
     3 |     1138.00 |     -132.00 |  319405.00 |        0.79
     4 |     1020.00 |      102.00 |  257499.00 |        1.22
     5 |     1047.00 |     -439.00 |  225872.00 |        0.41

Всего обработано элементов: 50000
```

Последовательный вариант ≈ параллельный (8 потоков) < параллельный (4 потока). По итогу этого задания самым быстрым оказался параллельный вариант на 8 потоков, запущенный локально.

Вывод

В ходе выполнения лабораторной работы были изучены основы параллельного программирования с использованием технологии OpenMP. Основные результаты:

1. **Реализованы параллельные версии** для следующих задач:
    - Суммирование элементов массива
    - Сортировка массива (быстрая сортировка)
    - Операции с матрицами (сложение, вычитание, умножение, деление)
2. **Проведено сравнение** производительности последовательных и параллельных версий программ:
    - Для малых объемов данных накладные расходы на создание потоков могут превышать выгоду от параллелизма
    - Наибольший прирост производительности наблюдается при обработке больших массивов данных
    - Оптимальное количество потоков зависит от характеристик вычислительной системы
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**:
    - Параллельные версии показывают лучшую производительность на больших объемах данных

Работа позволила получить практические навыки разработки параллельных приложений и анализа их эффективности.

Теоретическая справка

1. Основные понятия параллельного программирования

1.1 Параллелизм

Параллельное программирование - это подход к созданию программ, при котором несколько вычислительных процессов выполняются одновременно. В данной работе использовалась модель параллелизма с общей памятью, где несколько потоков имеют доступ к общему адресному пространству.

1.2 OpenMP

OpenMP (Open Multi-Processing) - э открытый стандарт для распараллеливания программ. В работе использовались следующие основные директивы OpenMP:

- #pragma omp parallel

- создание параллельной области

- #pragma omp parallel for

- распараллеливание циклов

- #pragma omp critical

- создание критической секции

- #pragma omp sections

- разделение кода на секции для параллельного выполнения

2. Управление потоками и памятью

2.1 Классы хранения переменных

- **private** - создает локальную копию переменной для каждого потока
- **shared** - переменная разделяется между всеми потоками
- **reduction** - выполняет указанную операцию над переменной из всех потоков

2.2 Планирование выполнения

Для эффективного распределения итераций цикла между потоками использовались:

- **static** - статическое распределение итераций

- **dynamic** - динамическое распределение итераций
- **chunk_size** - размер блока итераций для распределения

3. Алгоритм быстрой сортировки (QuickSort)

3.1 Основной принцип

Быстрая сортировка - это алгоритм "разделяй и властвуй", который:

1. Выбирает опорный элемент (pivot)
2. Разделяет массив на две части: элементы меньше опорного и больше опорного
3. Рекурсивно сортирует подмассивы

3.2 Параллельная реализация

В параллельной версии:

- Рекурсивные вызовы сортировки выполняются в разных потоках
- Используется директива

#pragma omp parallel sections

для параллельного выполнения рекурсивных вызовов

- Применяется порог параллелизма для оптимизации

3.3 Порог параллелизма

Порог параллелизма - это минимальный размер подмассива, при котором имеет смысл создавать новые потоки. Если размер подмассива меньше порога, используется последовательная сортировка. Это позволяет:

- Уменьшить накладные расходы на создание потоков
- Избежать избыточного распараллеливания
- Оптимизировать использование вычислительных ресурсов