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: Коллективные операции

Студент гр. 3381 _____ Козлов Г.Е.

Преподаватель _____ Татаринов Ю.С.

Санкт-Петербург
2025

Цель работы

Цель данной работы заключается в изучении и практическом применении механизмов группировки процессов и коллективных операций библиотеки MPI для решения задачи распределенного поиска минимума. Основная задача состоит в том, чтобы, используя функцию MPI_Comm_split, создать новый коммуникатор, включающий в себя только процессы с четным рангом в исходной группе, тем самым эффективно формируя подмножество участников. В рамках этого нового коммуникатора каждый процесс предоставляет набор из трех вещественных чисел. Применяя одну коллективную операцию редукции (MPI_Reduce) с оператором MPI_MIN, необходимо найти минимальное значение для каждого порядкового номера элемента (то есть покомпонентный минимум). Конечным результатом работы является вывод этих трех найденных минимальных значений в главном процессе (процессе с рангом 0 в исходном коммуникаторе), который также является корневым процессом в новом коммуникаторе.

Задание Вариант 4

В каждом процессе четного ранга (включая главный процесс) дан набор из трех элементов — вещественных чисел. Используя новый коммуникатор и одну коллективную операцию редукции, найти минимальные значения среди элементов исходных наборов с одним и тем же порядковым номером и вывести найденные минимумы в
главном процессе. Новый коммуникатор создать с помощью функции MPI_Comm_split. Указание. При вызове функции MPI_Comm_split в процессах, которые не требуется включать в новый коммуникатор, в качестве параметра color следует указывать константу MPI_UNDEFINED.

Выполнение работы.

Краткое описние алгоритма программы:

- Инициализация MPI:

Происходит вызов MPI_Init для запуска среды MPI.

Определяются ранг (rank) и общий размер (size) группы процессов в MPI_COMM_WORLD.

- Условная Инициализация Данных:

Процессы с четным рангом (rank % 2 == 0) инициализируют свой массив data (из трех элементов типа float) псевдослучайными значениями.

Процессы с нечетным рангом оставляют свой массив data с нулевыми значениями.

- Создание Нового Коммуникатора:

Определяется переменная color: 0 для четных рангов и MPI_UNDEFINED для нечетных.

Вызывается MPI_Comm_split для создания нового коммуникатора (newcomm), который включает только процессы, имеющие color = 0 (т.е., четные ранги).

- Коллективная Редукция:

Проверяется, создан ли newcomm (newcomm != MPI_COMM_NULL). Процессы, вошедшие в newcomm, вызывают MPI_Reduce.

Операция: MPI_MIN (поиск минимума).

Результаты (min_data) собираются и рассчитываются на корневом процессе (newrank = 0 в newcomm).

- Вывод и Завершение:

Корневой процесс в новом коммуникаторе (newrank == 0) выводит найденные покомпонентные минимальные значения.

Все процессы, создавшие newcomm, освобождают его память (MPI_Comm_free).

Вызывается MPI_Finalize для завершения работы среды MPI.

1. Формальное описание алгоритма

1. Листинг программы:

```
#include <mpi.h>
```

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(int argc, char** argv) {
    MPI_Init(&argc, &argv);

    int rank, size;
    MPI_Comm comm = MPI_COMM_WORLD;
    MPI_Comm_rank(comm, &rank);
    MPI_Comm_size(comm, &size);

    float data[3] = {0.0, 0.0, 0.0};
    if (rank % 2 == 0) {
        // Инициализация данных
        srand(time(NULL) + rank);
        data[0] = (float)(rand() % 100);
        data[1] = (float)(rand() % 100);
        data[2] = (float)(rand() % 100);
        printf("Процесс %d: %.2f %.2f %.2f\n", rank, data[0], data[1],
data[2]);
    }

    // Создание нового коммуникатора
    int color = (rank % 2 == 0) ? 0 : MPI_UNDEFINED;
    int key = rank;
    MPI_Comm newcomm;
    MPI_Comm_split(comm, color, key, &newcomm);

    if (newcomm != MPI_COMM_NULL) {
        int newrank;
        MPI_Comm_rank(newcomm, &newrank);

        float min_data[3];
        // Коллективная редукция для нахождения минимумов
        MPI_Reduce(data, min_data, 3, MPI_FLOAT, MPI_MIN, 0, newcomm);

        if (newrank == 0) {
            printf("Минимальные значения: %.2f %.2f %.2f\n",
min_data[0], min_data[1], min_data[2]);
        }
    }

    if (newcomm != MPI_COMM_NULL) {
        MPI_Comm_free(&newcomm);
    }

    MPI_Finalize();
    return 0;
}
```

Примеры запуска

```
george@archlinux ~/lab4_pa> mpirun -np 2 --oversubscribe ./a.out
Процесс 0: 43.00 86.00 74.00
Минимальные значения: 43.00 86.00 74.00
```

Рисунок 1

```
george@archlinux ~/lab4_pa> mpirun -np 12 --oversubscribe ./a.out
Процесс 4: 13.00 46.00 21.00
Процесс 8: 67.00 99.00 6.00
Процесс 10: 80.00 47.00 49.00
Процесс 6: 73.00 2.00 9.00
Процесс 2: 46.00 82.00 18.00
Процесс 0: 30.00 6.00 43.00
Минимальные значения: 13.00 2.00 6.00
```

1

Рисунок 3

2. Распечатка таблицы результатов и времени работы программы

Таблица 1 при n = 10000

Число процессов (p)	Время (сек) (Tp)	Ускорение (Sp=T1/Tp)	Эффективность (Ep=Sp/p) (%)
1	0.000015	1.00	100.00%
2	0.000032	0.47	23.44%
4	0.000022	0.68	17.05%
8	0.000129	0.12	1.45%
16	0.000748	0.02	0.13%

Таблица 2 при n = 50000

Число процессов (p)	Время (сек) (Tp)	Ускорение (Sp=T1/Tp)	Эффективность (Ep=Sp/p) (%)
1	0.000127	1.00	100.00%
2	0.000110	1.15	57.73%
4	0.000372	0.34	8.52%
8	0.000709	0.18	2.24%
16	0.000683	0.19	1.16%

Таблица 3 при n = 100000

Число процессов (p)	Время (сек) (Tp)	Ускорение (Sp=T1/Tp)	Эффективность (Ep=Sp/p) (%)
1	0.000165	1.00	100.00%
2	0.000184	0.90	44.88%
4	0.000473	0.35	8.71%
8	0.000725	0.23	2.84%
16	0.001356	0.12	0.76%

3. Графики времени выполнения программы в зависимости от объема исходных данных и числа запущенных процессов.

Рисунок 4

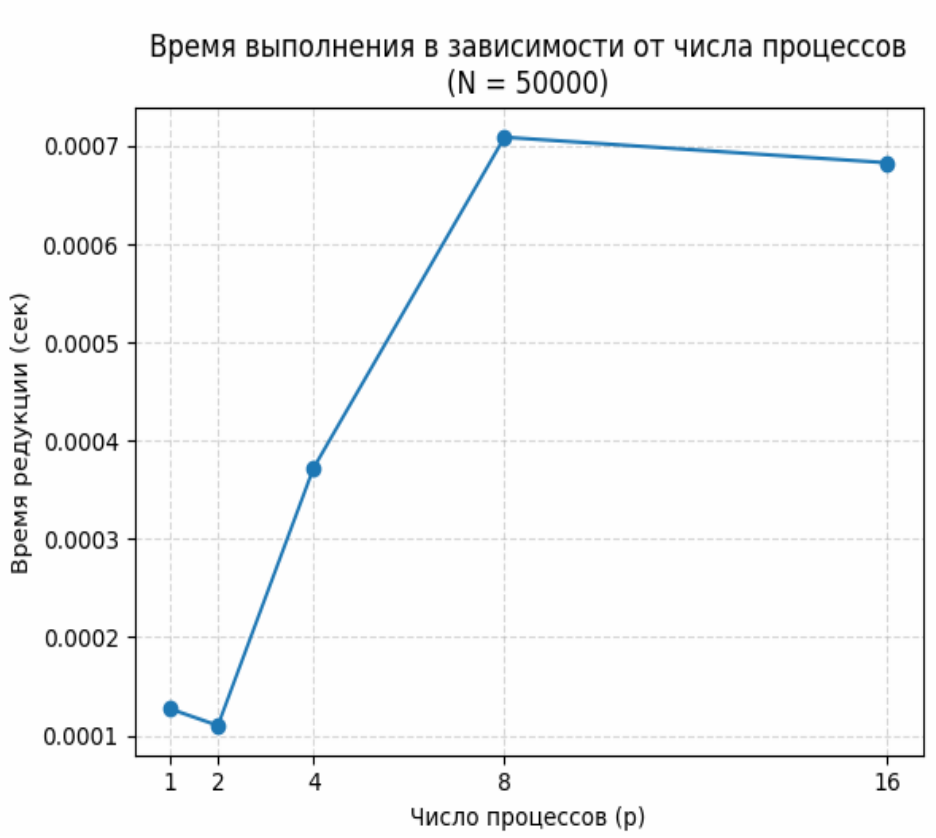

Рисунок 5

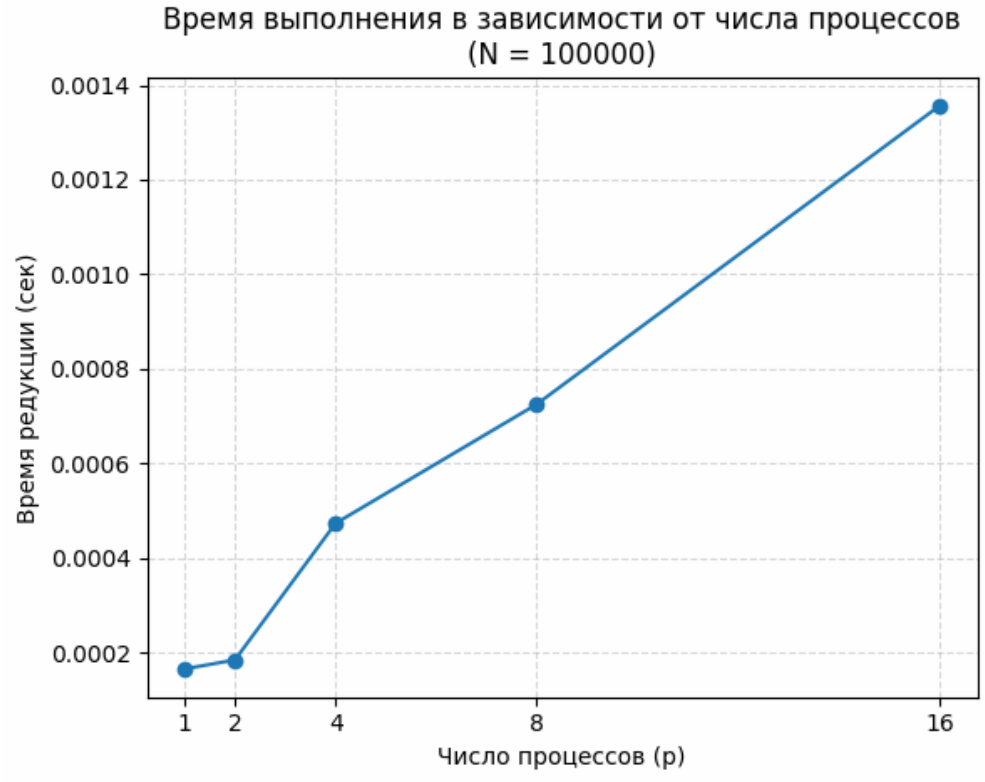

Рисунок 6

4. Графики ускорения на основании результатов, полученных выше.

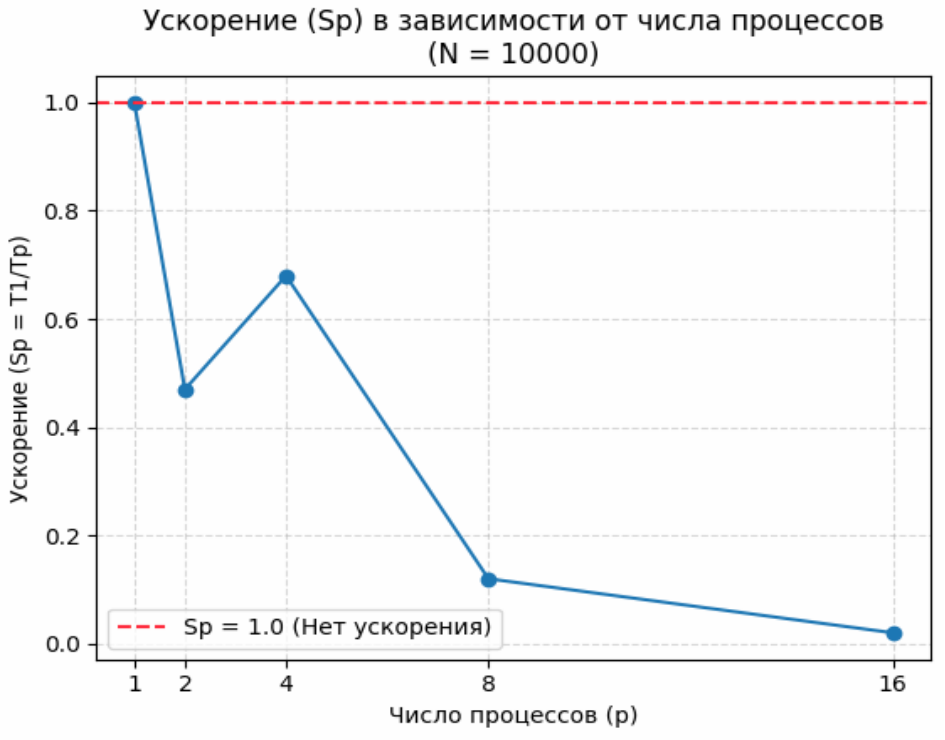

Рисунок 7

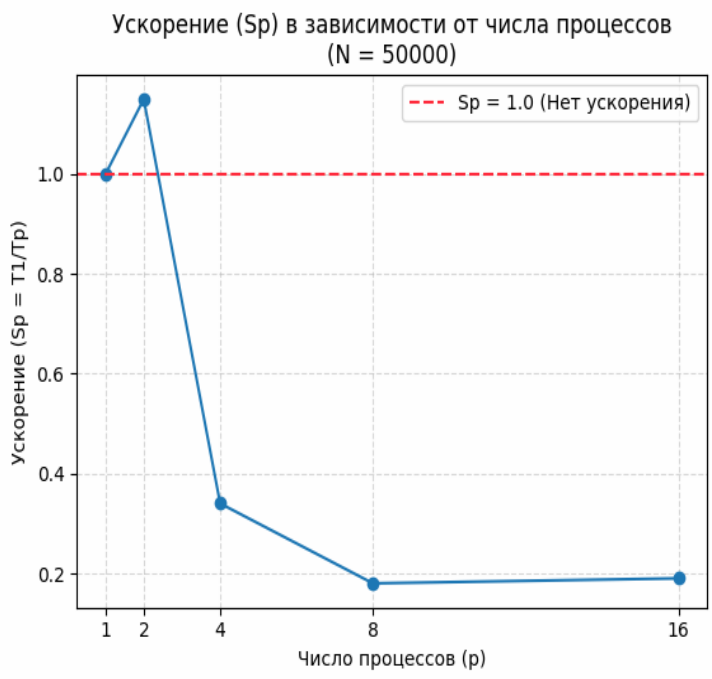

Рисунок 8

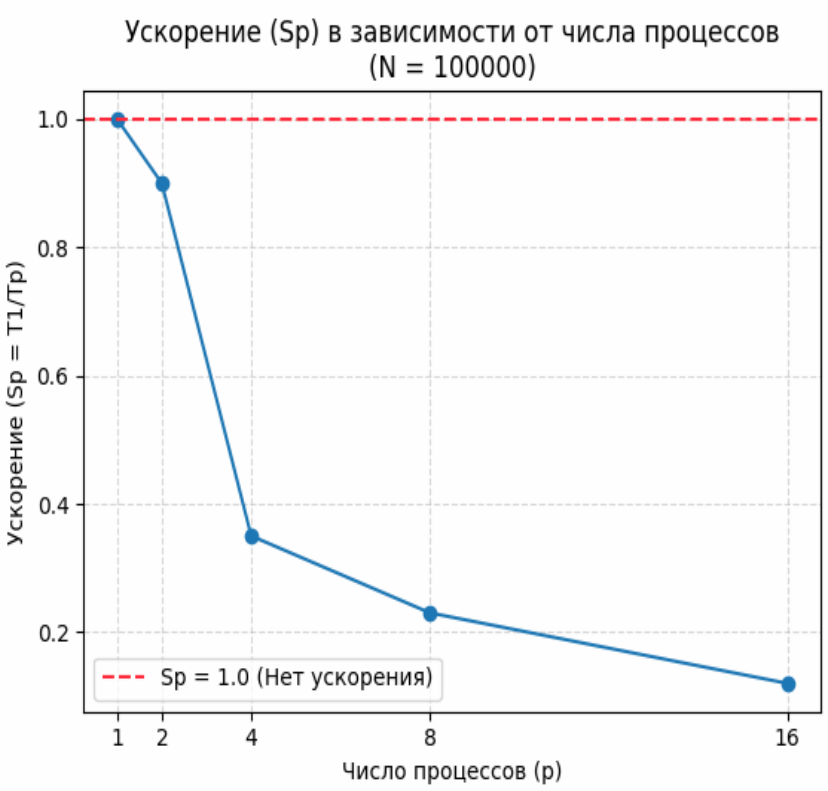

Рисунок 9

Вывод

Анализ графиков ускорения (Sp) и времени выполнения (Tp) показал, что для данной задачи, связанной с коллективной операцией редукции (`MPI_Reduce`) над малым объемом данных, **параллелизация является неэффективной и приводит к замедлению программы** во всех исследованных случаях, кроме одного исключения. Основной причиной такого поведения является **доминирование накладных расходов MPI** (связанных с синхронизацией и коммуникацией) над временем полезных вычислений. Поскольку задача, вероятно, представляет собой многократное выполнение тривиальной операции редукции над тремя элементами, время, необходимое на вычисление, чрезвычайно мало. На графиках ускорения (Sp) для N=10000 и N=100000 наблюдается **монотонное падение ускорения** до значений, близких к нулю, сразу после p=1 или p=2, что означает, что программа начинает работать медленнее, чем последовательный вариант. **Точка перелома** (где ускорение падает ниже 1.0) находится либо на p=1, либо на p=2, демонстрируя, что уже при небольшом увеличении числа процессов накладные расходы становятся

неприемлемыми. Единственное незначительное ускорение (Sp≈1.15) наблюдалось при N=50000 и p=2, но даже в этом случае резкий рост времени выполнения при p=4 показал, что масштабируемость отсутствует. Увеличение размерности задачи (параметра N до 100000) **не смогло компенсировать** эти накладные расходы, поскольку время T1 (последовательное выполнение) увеличилось, но накладные расходы на коммуникацию росли быстрее с увеличением числа процессов, что привело к наихудшему ускорению (Sp≈0.12 при p=16) именно для самого большого объема данных. Таким образом, эксперимент наглядно доказывает, что **задачи с низкой вычислительной интенсивностью не подходят для распараллеливания средствами MPI**, так как накладные расходы коммуникации сводят на нет любой потенциальный выигрыш.