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: Группы процессов и коммуникаторы.

Студент гр. 3388 _____ Дубровин Д.Н.

Преподаватель _____ Татаринов Ю.С.

Санкт-Петербург

2025

Цель работы.

Целью данной работы является освоение механизма работы с коммуникаторами в MPI и коллективными операциями для организации групповых вычислений в параллельных приложениях.

Задание.

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.

1. Программа инициализирует MPI, определяя ранг и размер коммуникатора. Замеряется время начала выполнения. Генерируется массив из трех случайных чисел с использованием различного seed для каждого процесса для обеспечения уникальности данных. Выводятся сгенерированные данные каждого процесса. Создается новый коммуникатор для процессов с четным рангом через MPI_Comm_split, где процессы с нечетным рангом исключаются через MPI_UNDEFINED. В новом коммуникаторе выполняется операция редукции MPI_Reduce с операцией MPI_MIN для поиска минимальных значений по каждому из трех элементов. Результат собирается в процессе с рангом 0 нового коммуникатора. Главный процесс выводит найденные минимумы, общее количество

процессов и время выполнения. Происходит освобождение коммуникатора и финализация MPI.

2. Построим схему Петри

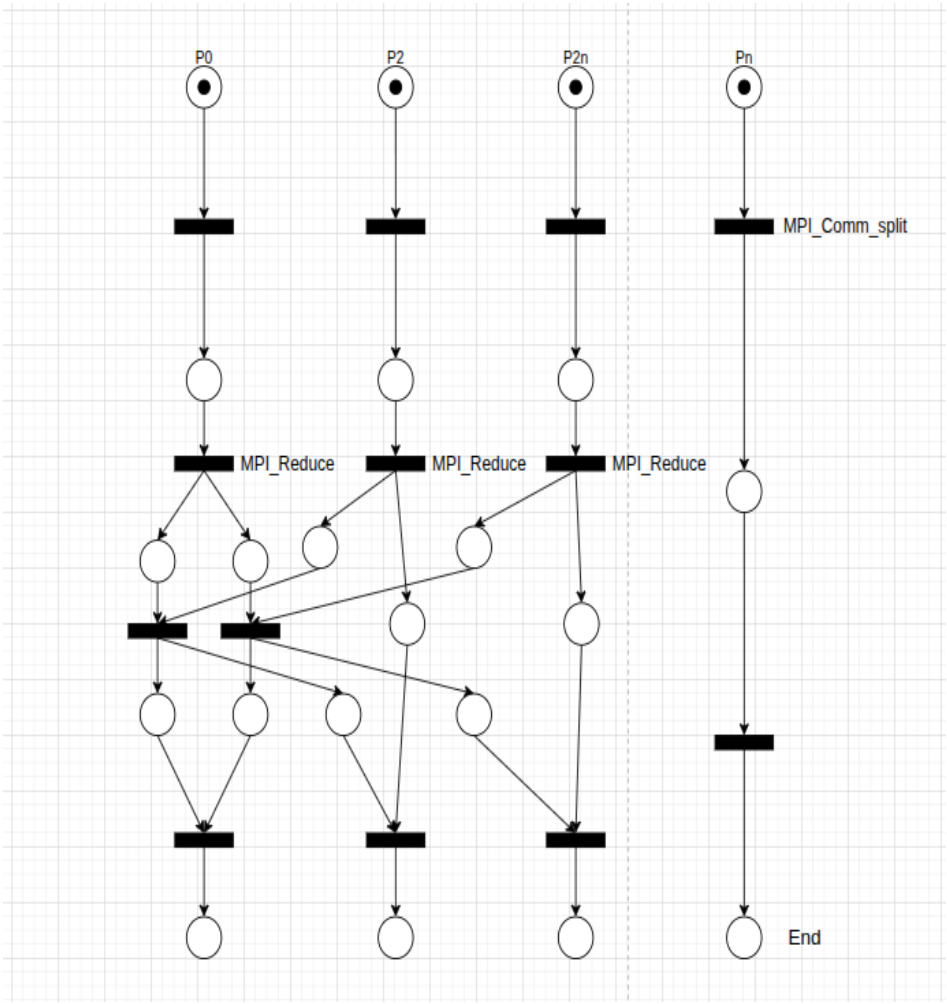

Рис.1 - сеть Петри

```
rendex@ThinkPad-T490 ~/Repos/parallel_algorithms/lab4 (main)
❯ mpirun --oversubscribe -np 5 ./task
Process 1: data = [92.30, 50.30, 26.20]
Process 2: data = [51.90, 71.50, 0.30]
Process 0: data = [84.00, 27.40, 99.90]
Process 3: data = [47.00, 53.00, 14.20]
Process 4: data = [48.40, 14.60, 9.40]
============= Execution Results =============
Min values: [48.40, 14.60, 0.30]
Num of proc: 5 | Execution time: 0.000438 seconds⏎
```

Рис.2 — пример работы программы

3. Проведём запуск при разном колличестве процессов.

Таблица 1- результаты выполнения программы при различном количестве процессов.

N	1	3	5	7	9	11	13	15
t, сек	0.000142	0.000334	0.000544	0.000405	0.000633	0.015152	0.003728	0.004915

4. Построим графики времени выполнения и ускорения для 1-15 процессов.

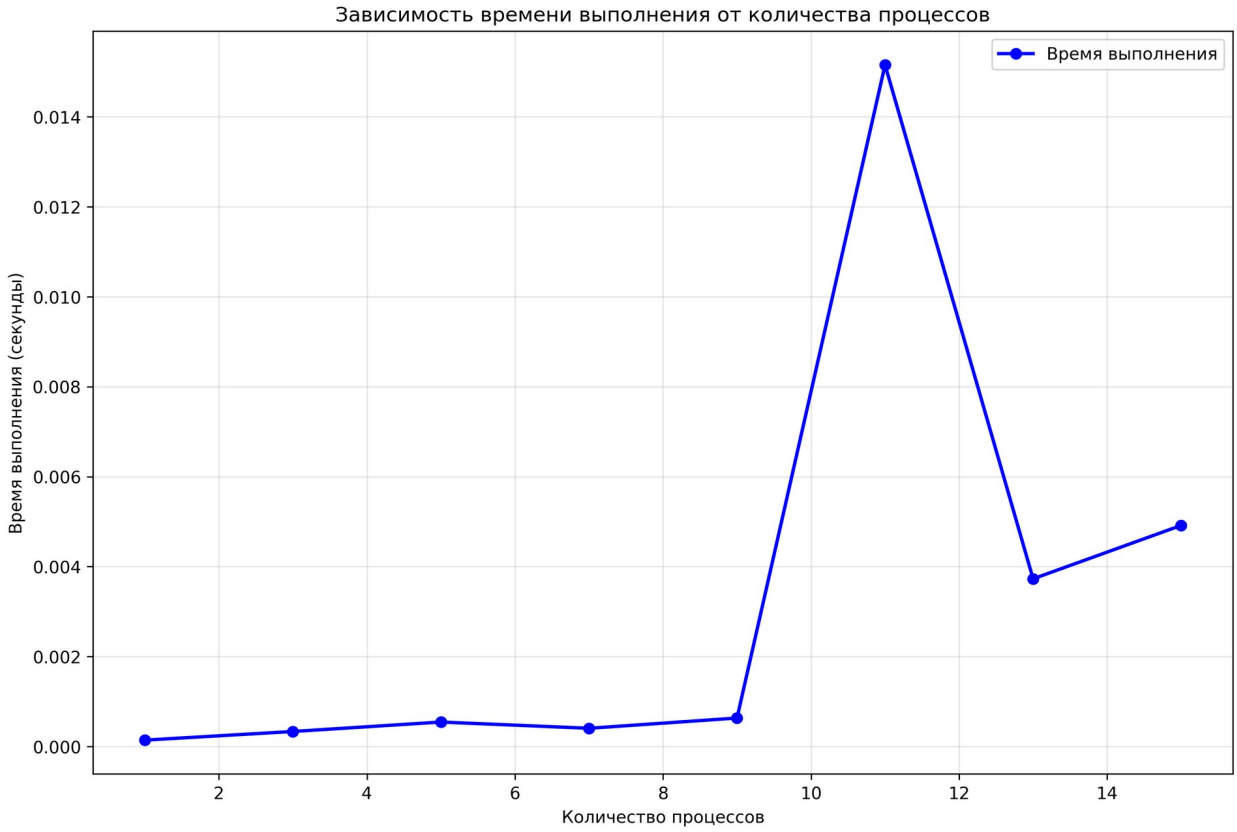

Рис.3 – график зависимости времени выполнения от числа процессов

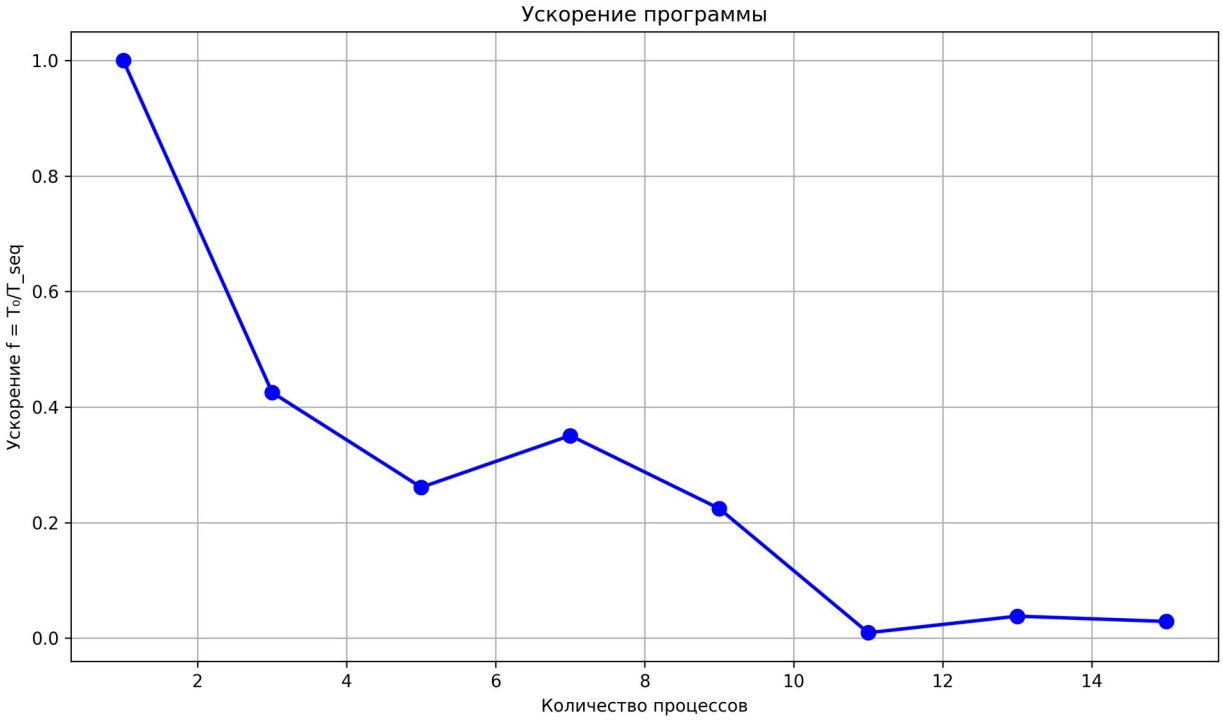

Рис.4 - график ускорения

Анализ результатов.

Ожидаемо, с ростом входных данных (количества массивов) программа демонстрирует замедление. Однако этот рост оптимизируется засчёт распараллеливания программы.

Разработанный программный код см. в приложении А.

Выводы.

В ходе работы успешно реализован алгоритм, использующий функцию MPI_Comm_split для разделения процессов на группы. Продемонстрирована эффективность операции MPI_Reduce с параметром MPI_MIN для поиска минимальных значений в распределенных массивах.

Приложение А

Исходный код программы

Название файла: task.c

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

int main(int argc, char **argv) {
    int world_rank, world_size;
    double start_time, end_time;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    start_time = MPI_Wtime();

    double data[3];
    srand(time(NULL) + world_rank);
    for (int i = 0; i < 3; ++i) {
        data[i] = (double)(rand() % 1000) / 10.0;
    }

    printf("Process %d: data = [%.2f, %.2f, %.2f]\n",
        world_rank, data[0], data[1], data[2]);

    MPI_Comm even_comm;
    int color = ((world_rank % 2 == 0) ? 0 : MPI_UNDEFINED);

    MPI_Comm_split(MPI_COMM_WORLD, color, world_rank, &even_comm);

    if (color != MPI_UNDEFINED) {
        int even_rank, even_size;

        MPI_Comm_rank(even_comm, &even_rank);
        MPI_Comm_size(even_comm, &even_size);

        double min_values[3];
```

```
                    MPI_Reduce(data, min_values, 3, MPI_DOUBLE, MPI_MIN, 0,
even_comm);

                if (even_rank == 0) {
                    end_time = MPI_Wtime();

                    puts("============= Execution Results =============");
                    printf("Min values: [%.2f, %.2f, %.2f]\n",
                        min_values[0], min_values[1], min_values[2]);
                    printf("Num of proc: %d | Execution time: %.6f seconds",
                        world_size, end_time - start_time);
                }

                MPI_Comm_free(&even_comm);
            }

            MPI_Finalize();
            return 0;
        }
```