МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №5

по дисциплине «Параллельные алгоритмы»

Тема: Виртуальные топологии.

Студент гр. 3388 _____ Дубровин Д.Н.

Преподаватель _____ Татаринов Ю.С.

Санкт-Петербург

2025

Цель работы.

Целью данной работы является изучение виртуальных топологий в MPI и их применение на практике.

Задание.

6. Число процессов K кратно трем: K = 3N, N > 1 В процессах 0, 1 и 2 дано по N целых чисел. Определить для всех процессов декартову топологию в виде матрицы размера N × 3, после чего, используя функцию MPI_Cart_sub, расщепить матрицу процессов на три одномерных столбца (при этом процессы 0, 1 и 2 будут главными процессами в полученных столбцах). Используя одну коллективную операцию пересылки данных, переслать по одному исходному числу из главного процесса каждого столбца во все процессы этого же столбца и вывести полученное число в каждом процессе (включая процессы 0, 1 и 2).

Выполнение работы.

1. Программа на языке C с использованием MPI организует заданное количество процессов в двумерную декартову решетку размером N на 3. Процессы, находящиеся в первой строке этой решетки, инициализируют числовые данные (100, 200 и 300) для каждого из трех столбцов. Затем, с помощью коллективной операции широковещательной рассылки (MPI_Bcast), эти данные передаются всем остальным процессам в пределах того же столбца. После получения данных каждый процесс выводит на экран свой глобальный ранг, координаты в решетке и полученное значение. В самом конце главный процесс (с рангом 0) измеряет и печатает общее время выполнения всей программы.

2. Построим схему Петри

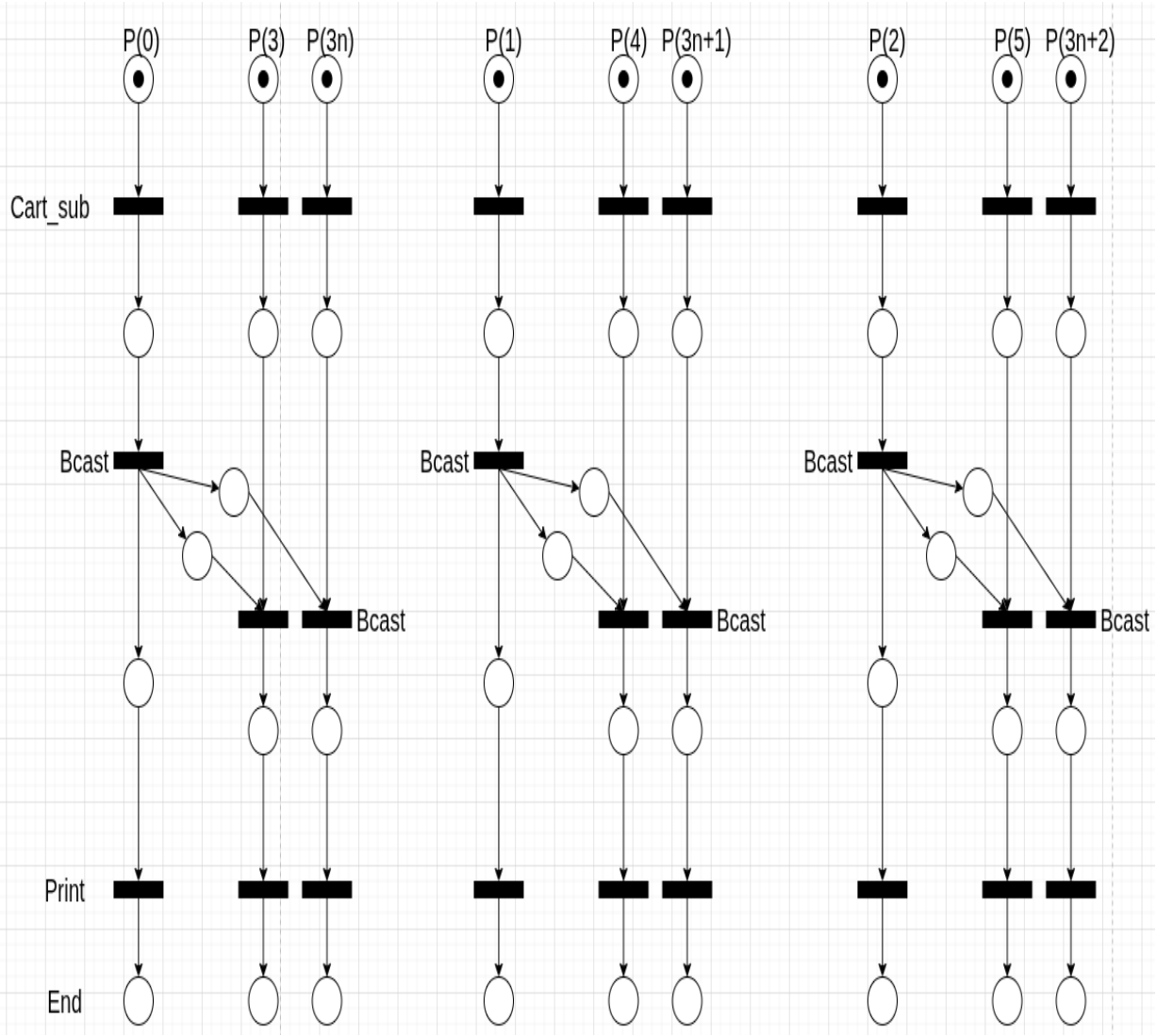

Рис.1 - сеть Петри

```
❯ mpirun --oversubscribe -np 6 ./task
Process 2 (coords [0,2]) received data: 300
Process 1 (coords [0,1]) received data: 200
Process 0 (coords [0,0]) received data: 100
Process 4 (coords [1,1]) received data: 200
Process 5 (coords [1,2]) received data: 300
Process 3 (coords [1,0]) received data: 100
Num of proc: 6 | Execution time: 0.000253 seconds
```

Рис.2 — пример работы программы

3. Проведём запуск при разном колличестве процессов.

Таблица 1- результаты выполнения программы при различном количестве процессов.

N	3	6	9	12	15	18	21	24
t, сек	0.000168	0.000274	0.000356	0.000440	0.000498	0.000960	0.001045	0.001885

4. Построим графики времени выполнения и ускорения для 3-24 процессов.

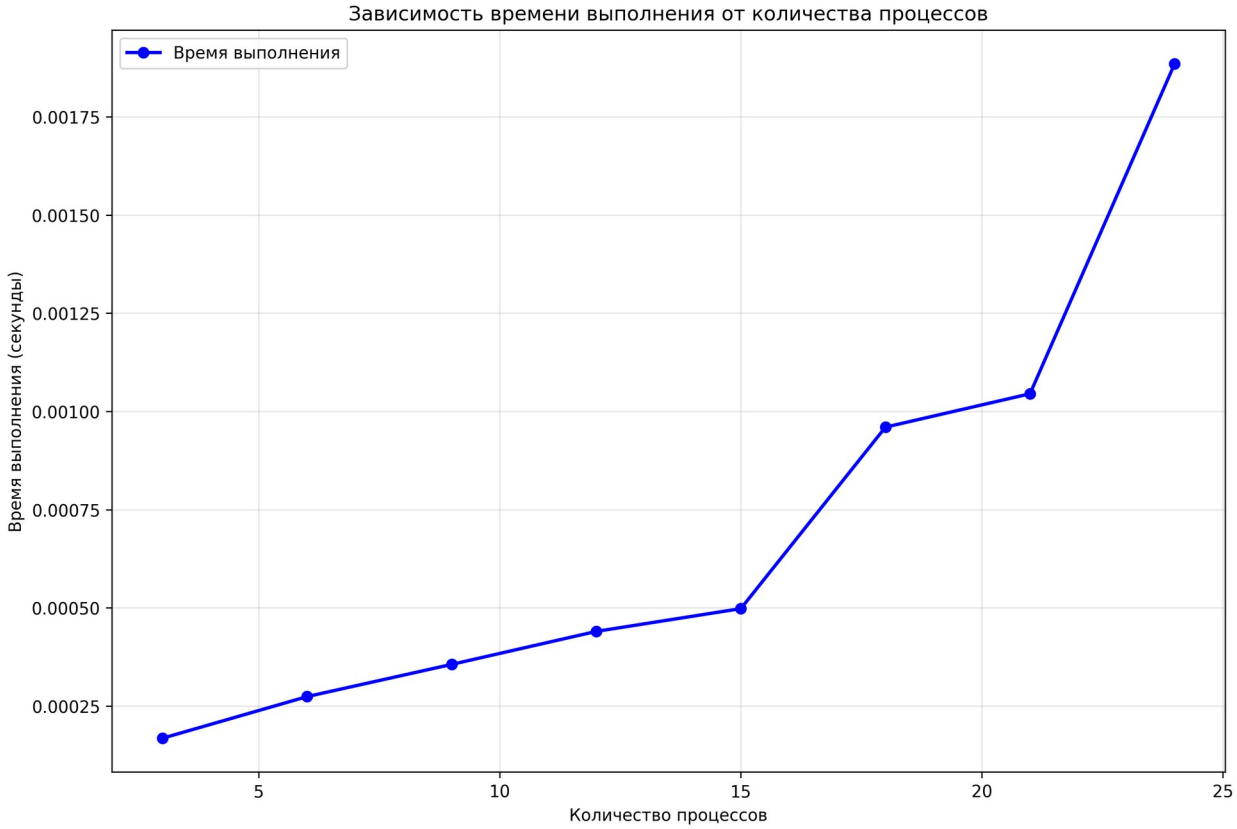

Рис.3 – график зависимости времени выполнения от числа процессов

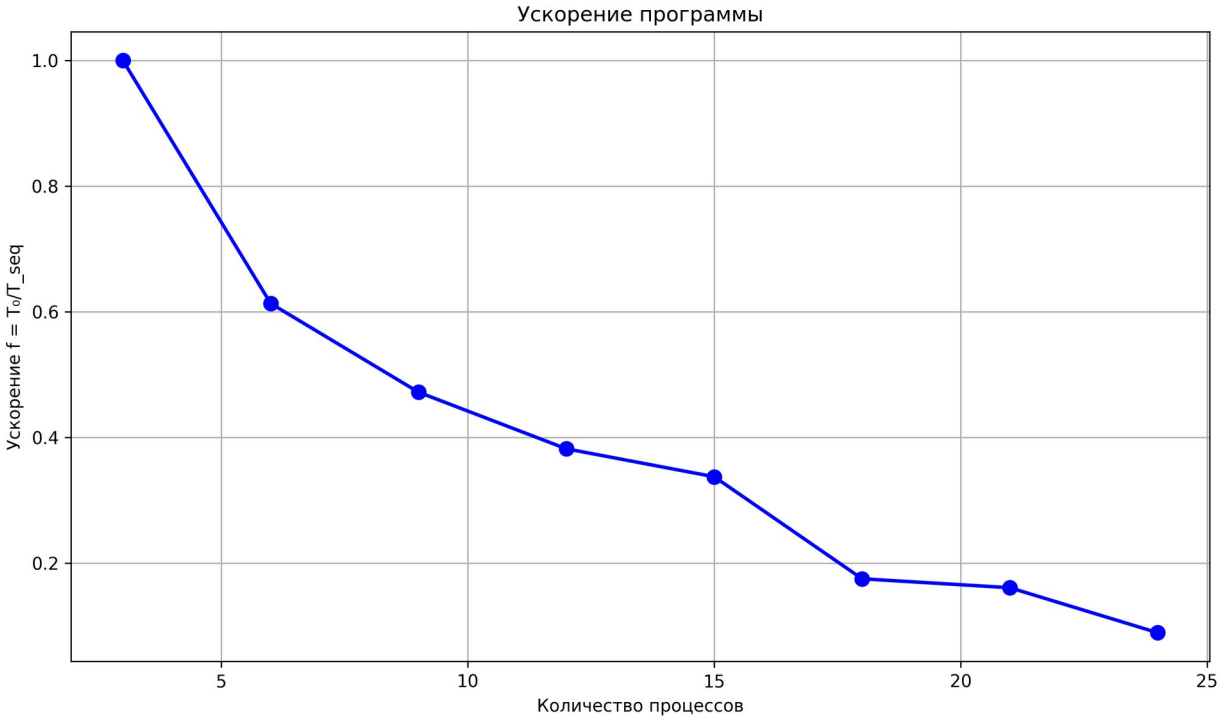

Рис.4 - график ускорения

Анализ результатов.

Ожидаемо, с ростом входных данных (количества массивов) программа демонстрирует замедление. Однако этот рост оптимизируется засчёт распараллеливания программы. Интересно, что, когда количество процессов превышает количество физических ядер на машине (16), наблюдается резкий скачок времени выполнения программы, что объясняется возникающим псевдопараллелизмом.

Разработанный программный код см. в приложении А.

Выводы.

В ходе работы были освоены ключевые функции MPI для работы с декартовыми топологиями. На практике было показано, как разделение процессов на логические подгруппы (столбцы) позволяет эффективно локализовать коллективные операции, делая код более структурированным и масштабируемым.

Приложение А

Исходный код программы

Название файла: task.c

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

int main(int argc, char *argv[])
{
    int global_rank, global_size;
    int N;
    int dims[2], periods[2], coords[2];
    int remain_dims[2];
    int bcast_data;
    double start_time, end_time;
    MPI_Comm cart_comm, column_comm;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &global_rank);
    MPI_Comm_size(MPI_COMM_WORLD, &global_size);

    if (global_rank == 0) {
        start_time = MPI_Wtime();
    }

    if (global_size % 3 != 0) {
        if (global_rank == 0) {
                printf("Error: Number of processes must be multiple of
3\n");
        }
        MPI_Finalize();
        return 1;
    }

    N = global_size / 3;

    dims[0] = N;
    dims[1] = 3;
    periods[0] = 0;
```

```
    periods[1] = 0;

    int cart_rank;
    MPI_Cart_create(MPI_COMM_WORLD, 2, dims, periods, 1, &cart_comm);
    MPI_Comm_rank(cart_comm, &cart_rank);
    MPI_Cart_coords(cart_comm, cart_rank, 2, coords);

    if (coords[0] == 0) {
        bcast_data = 100 * (coords[1] + 1);
    }

    remain_dims[0] = 1;
    remain_dims[1] = 0;

    MPI_Cart_sub(cart_comm, remain_dims, &column_comm);

    MPI_Bcast(&bcast_data, 1, MPI_INT, 0, column_comm);

    printf("Process %d (coords [%d,%d]) received data: %d\n",
           global_rank, coords[0], coords[1], bcast_data);

    MPI_Comm_free(&column_comm);
    MPI_Comm_free(&cart_comm);

    MPI_Barrier(MPI_COMM_WORLD);

    if (global_rank == 0) {
        end_time = MPI_Wtime();
        printf("Num of proc: %d | Execution time: %.6f seconds",
                global_size, end_time - start_time);
    }

    MPI_Finalize();

    return 0;
}
```