

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

ОТЧЕТ
по лабораторной работе №5
по дисциплине «Параллельные алгоритмы»
ТЕМА: Виртуальные топологии.

Студент

Степаненко Д. В.

Преподаватель

Татаринов Ю. С.

Санкт-Петербург

2023 г.

Цель

Исследовать виртуальные топологии в библиотеки MPI, с использованием функций для ее создания, а также рассмотреть операции с топологиями, в частности сдвиг.

Постановка задачи (вариант 10)

Число процессов K является четным: $K = 2N$, $N > 1$; в каждом процессе дано вещественное число A . Определить для всех процессов декартову топологию в виде матрицы размера $2 \times N$ (порядок нумерации процессов оставить прежним) и для каждой строки матрицы осуществить циклический сдвиг исходных данных с шагом 1 (число A из каждого процесса, кроме последнего в строке, пересылается в следующий процесс этой же строки, а из последнего процесса — в главный процесс этой строки). Для определения рангов посылающих и принимающих процессов использовать функцию `MPI_Cart_shift`, пересылку выполнять с помощью функции `MPI_Sendrecv`. Во всех процессах вывести полученные данные.

Выполнение работы

Первоначально происходит инициализация MPI, определяются общее количество процессов и ранг текущего процесса. Затем определяются значение переменной N - количество строк в декартовой топологии (количество всех процессов/2). Переменная A принимает значение ранга текущего процесса.

Далее происходит создание декартовой топологии с помощью функции `MPI_Cart_create()`. После определяются переменные `recv_data`, `source` и `destination`. Переменная `recv_data` будет хранить принятые данные от другого процесса при сдвиге, а переменные `source` и `destination` - ранг исходного и целевого процессов при передаче данных.

Далее вызывается функция `MPI_Cart_shift()`, которая определяет источник и цель передачи данных в декартовой топологии. Вызывается функция `MPI_Sendrecv()`, которая осуществляет пересылку своего ранга вдоль оси *y* к соответствующему процессу и получает ранг от другого процесса. Под конец освобождается коммуникатор, выводится информация о полученных данных с помощью функции, и освобождаются ресурсы системы.

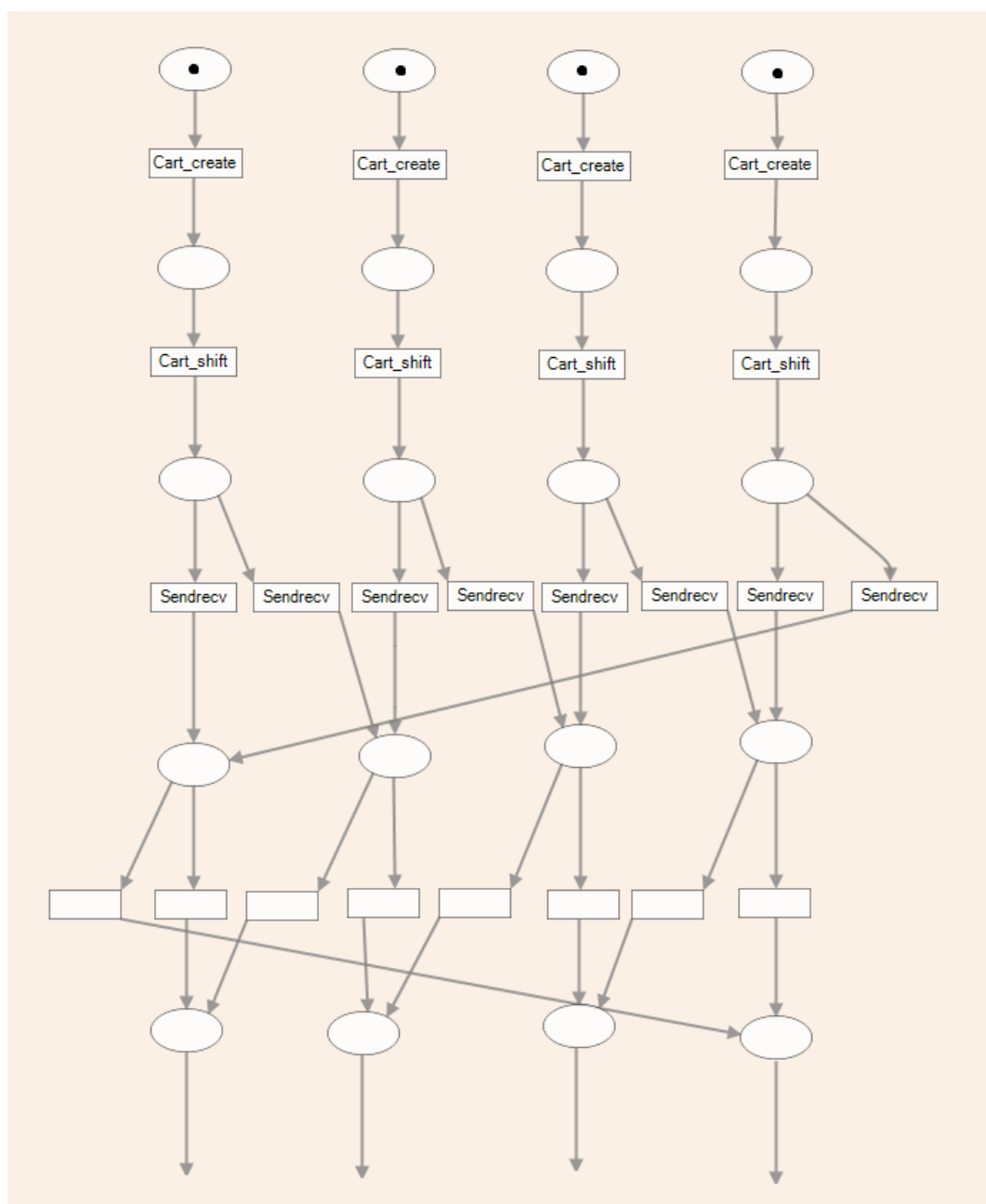


Рисунок 1 - Сеть Петри основной параллельной части программы для четырех процессов.

Листинг программы:

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

int main(int argc, char** argv) {
    int size, rank;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    int N = size / 2;
    int A = rank;

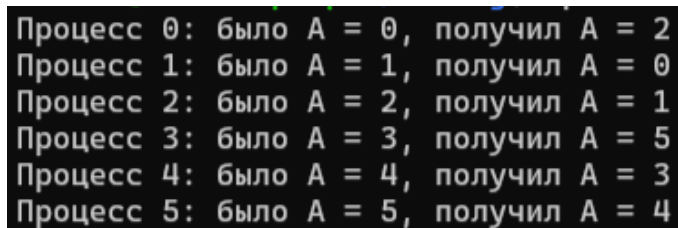
    // Создание декартовой топологии
    int dims[2] = {2, N};
    int periods[2] = {0, 1}; //для передачи по строке от последнего к
главному
    int reorder = 0;
    MPI_Comm cart_comm;
    // Аргументы: коммуникатор; размерность; массив размеров сетки;
массив периодов; переупорядочивание; новый коммуникатор;
    MPI_Cart_create(MPI_COMM_WORLD, 2, dims, periods, reorder,
    &cart_comm);

    int recv_data;
    int source;
    int destination;

    // Аргументы: коммуникатор; направление 0 - вверх, 1 - по y;
размер смещения; источник; назначение;
    MPI_Cart_shift(cart_comm, 1, 1, &source, &destination);
    MPI_Sendrecv(&A, 1, MPI_INT, destination, 0, &recv_data, 1,
MPI_INT, source, 0, cart_comm, MPI_STATUS_IGNORE);
    MPI_Comm_free(&cart_comm);

    printf("Процесс %d: было A = %d, получил A = %d\n", rank, A,
recv_data);

    MPI_Finalize();
    return 0;
}
```

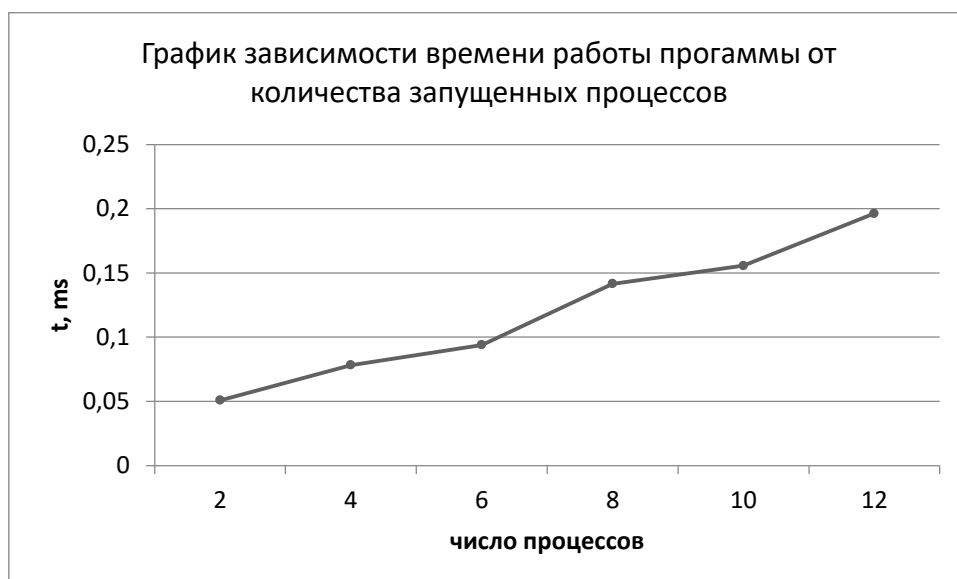


Процесс 0: было A = 0, получил A = 2
Процесс 1: было A = 1, получил A = 0
Процесс 2: было A = 2, получил A = 1
Процесс 3: было A = 3, получил A = 5
Процесс 4: было A = 4, получил A = 3
Процесс 5: было A = 5, получил A = 4

Рисунок 2 - Полученный вывод при запуске программы на шести процессах.

Таблица 1 – Результаты работы программы на разном количестве процессов.

Количество процессов (шт)	Среднее затрачиваемое время (мс)
2	0,0508
4	0,0782
6	0,0939
8	0,1416
10	0,1556
12	0,1963



Расчеты ускорения программы выполним по формуле:

$$S_p(n) = T_1(n)/T_p(n)$$

Таблица 2 – Результаты расчетов ускорения программы.

Количество процессов P (шт)	Ускорение S_p
2	1
4	0,65
6	0,541

Количество процессов P (шт)	Ускорение S_p
8	0,359
10	0,326
12	0,259



Вывод

В ходе выполнения лабораторной работы были изучены операции с созданием и работой с топологиями в библиотеке MPI, использованы на практике функции *MPI_Cart_create()* и *MPI_Cart_shift()*. Первая – создает новую топологию по заданным параметрам (размерность, периодичность, перемешивание) на основе нового коммутатора. Вторая – возвращает координаты, которые можно будет использовать для циклического сдвига. Также имеет настраиваемые параметры: направление и размер смещения. Для пересылки сообщений по заданным координатам использовалась блокируемая функция *MPI_Sendrecv()*. Таким образом, была написана программа, удовлетворяющая ТЗ.

Время выполнения программы засекалось на параллельной части кода, где использовались вышеперечисленные функции. Количество процессов

изменялось от 1 до 12, т.к. дальнейшее увеличение было бы малоинформативным. На рисунке 1 видно, что скорость выполнения программы увеличивается линейно. Это происходит потому, что с добавлением новой пары процессов длина строки (по оси y) увеличивается на 1. Также количество процессов N влияет на скорость создания новой топологии. Следовательно, скорость исполнения программного кода будет пропорционально количеству процессов.

Исходя из времени выполнения программы, можно сделать вывод, что и ускорение будет уменьшаться с увеличением процессов. Эту зависимость можно посмотреть на графике.