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

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Системы параллельной обработки данных»

Тема: Виртуальные топологии

Студент гр. 1310 _____ Комаров Д.Е.

Преподаватель _____ Татаринов Ю.С.

Санкт-Петербург

2025

Цель работы

Целью выполнения лабораторной работы является построение программы с использованием виртуальных топологий при помощи библиотеки MPI.

Постановка задачи

Вариант 9. В каждом подчиненном процессе дано вещественное число. Определить для всех процессов декартову топологию в виде одномерной решетки и осуществить простой сдвиг исходных данных с шагом −1 (число из каждого *подчиненного* процесса пересылается в предыдущий процесс). Для определения рангов посылающих и принимающих процессов использовать функцию MPI_Cart_shift, пересылку выполнять с помощью функций MPI_Send и MPI_Recv. Во всех процессах, получивших данные, вывести эти данные.

Выполнение работы

Программа, выполняющая поставленную задачу представлена в приложении А. Выполнение программы начинается с определения ранга текущего процесса и количества процессов. После чего создается виртуальная топология в виде одномерной решетки с размером одного измерения, равному количеству процессов. Затем в каждом процессе генерируется вещественное число, равное рангу процесса, умноженному на 0.01. После этого при помощи MPI_Cart_shift определяются ранги процессов для выполнения циклического сдвига с шагом -1. Далее сгенерированное вещественное число пересылается предыдущему процессу при помощи MPI_Send и получается от следующего процесса при помощи MPI_Recv. Каждый процесс, получивший число выводит его на экран. На рисунке 1 представлен результат работы программы для 4-х процессов.

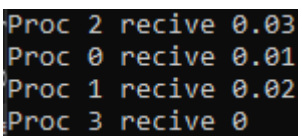

Рисунок 1 – Демонстрация работы программы

На рисунке 2 представлен алгоритм программы в виде сети Пери для 4-х процессов.

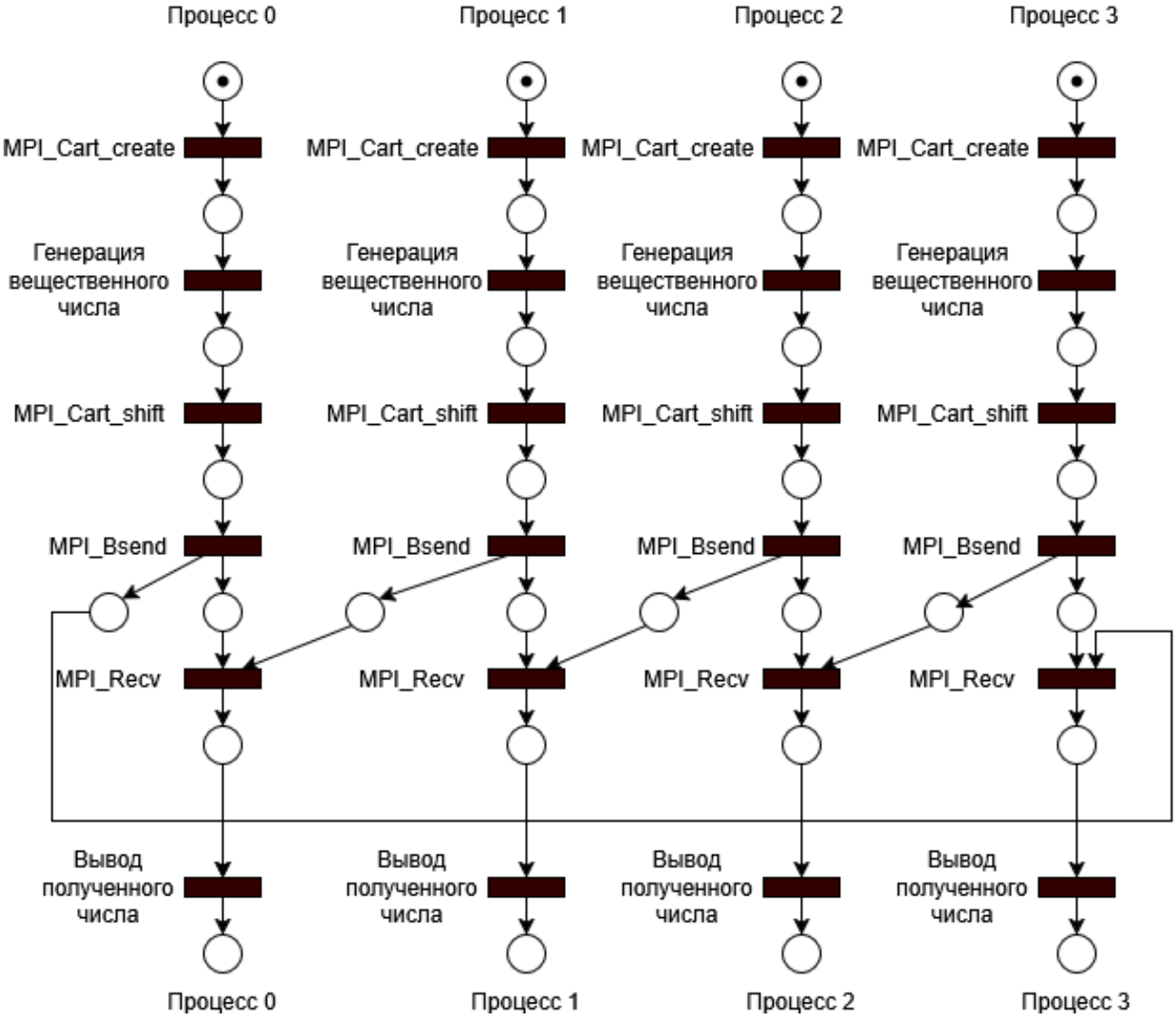

Рисунок 2 – Сеть Петри

Протестируем полученный алгоритм на различном количестве процессов и различном количестве пересылаемых чисел. В измерения не включено время подготовки данных и вывода полученных данных на экран.

Время работы алгоритма при пересылке 1 числа представлено в таблице 1.

Таблица 1 – Время работы алгоритма при пересылке 1 числа

Число процессов	Время выполнения (мс)
2	0.01

Продолжение таблицы 1

4	0.13
8	0.34
16	1.22
32	1.7

Время работы алгоритма при пересылке 100 чисел представлено в таблице 2.

Таблица 2 – Время работы алгоритма при пересылке 100 чисел

Число процессов	Время выполнения (мс)
2	0.01
4	0.1
8	0.22
16	0.32
32	4.77

Время работы алгоритма при пересылке 1000 чисел представлено в таблице 3.

Таблица 3 – Время работы алгоритма при пересылке 1000 чисел

Число процессов	Время выполнения (мс)
2	0.01
4	0.13
8	0.18
16	0.79
32	3.6

На рисунке 3 представлен график, отражающий зависимость времени выполнения от количества процессов.

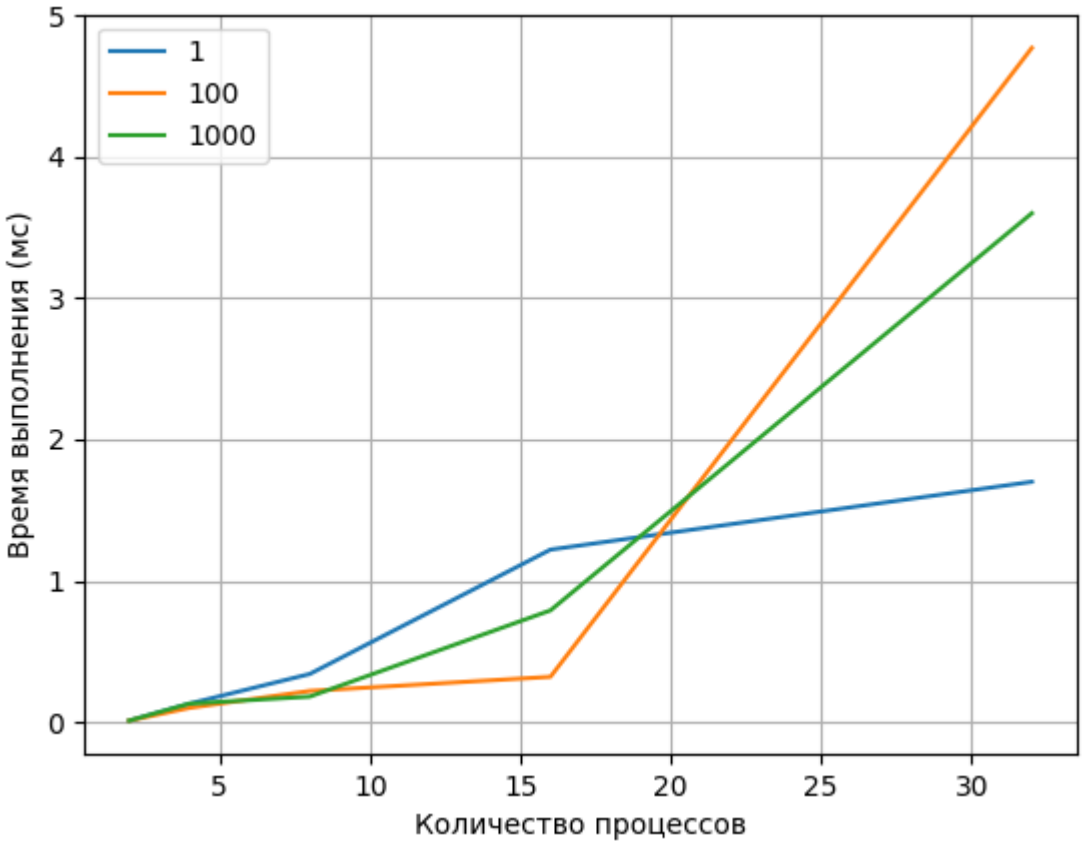

Рисунок 3 – График, отражающий зависимость времени выполнения от количества процессов

На рисунке 4 представлен график, отражающий зависимость времени выполнения от количества пересылаемых чисел.

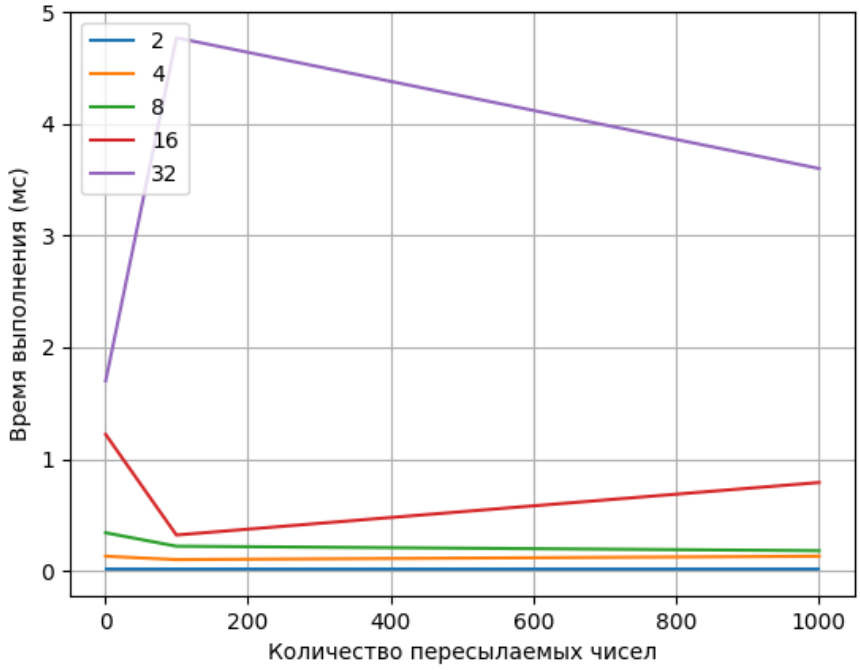

Рисунок 4 – График, отражающий зависимость времени выполнения от количества пересылаемых чисел

Из таблиц 1–3 и рисунков 3 и 4 можно заметить, что время выполнения растет с увеличением количества процессов, однако для большого количества процессов время выполнения имеет случайный характер. Объясняется это тем, что то, как операционная система переключается между процессами так же случайно. Может быть как оптимальный случай, когда ОС переключается между процессами в том же порядке, что они и пересылают данные друг другу, так и худший случай, когда переключения между процессами будут в таком порядке, что большая часть процессов будет ожидать какой-либо процесс, переключение к которому будет осуществлено в последнюю очередь. Рост времени выполнения программы с увеличением количества пересылаемых чисел замечен не был, что объясняется высокой эффективностью пересылки данных между процессами.

На рисунке 5 представлен график замедления выполнения программы в зависимости от количества процессов.

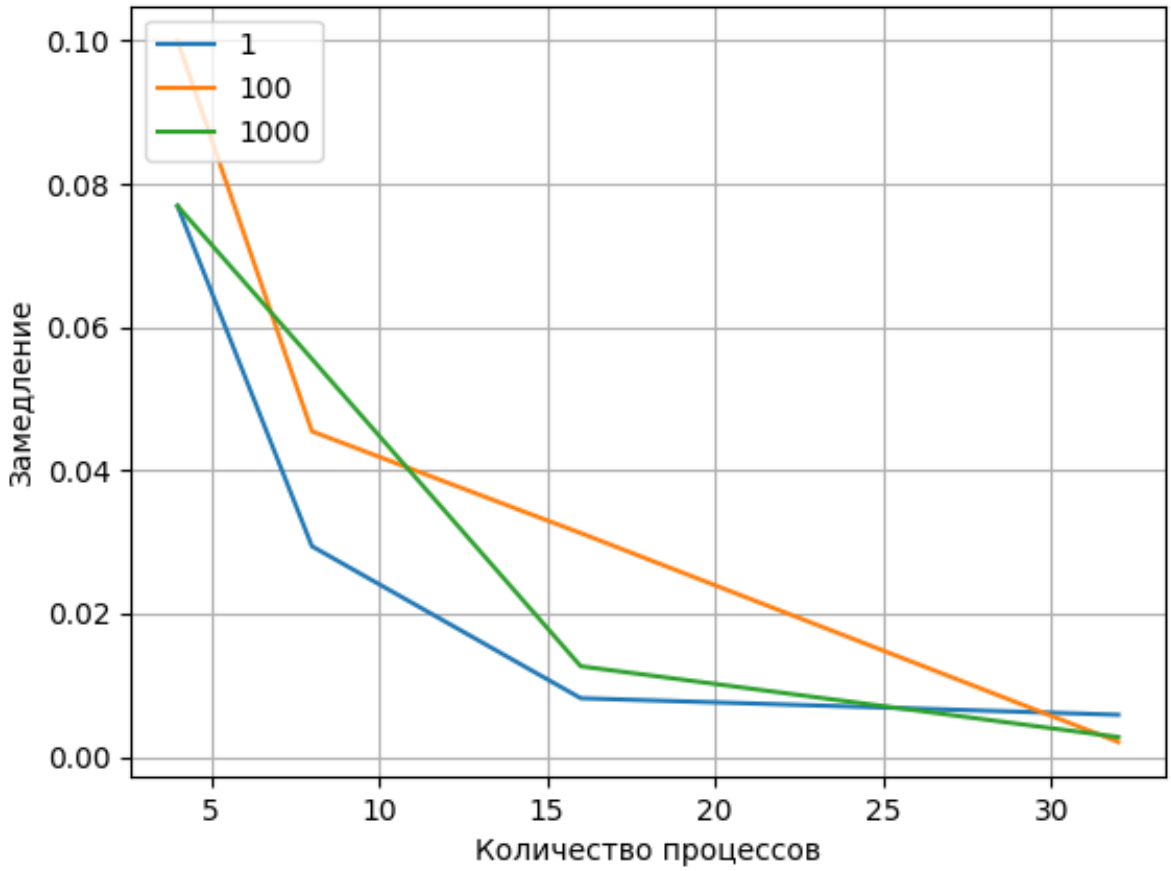

Рисунок 5 – График замедления выполнения программы в зависимости от количества процессов

Выводы

В ходе выполнения лабораторной работы была написана программа с использованием виртуальных топологий при помощи библиотеки MPI.

Было выяснено, что для данной программы время выполнения растет с увеличением количества процессов, однако для большого количества процессов время выполнения имеет случайный характер. Объясняется это тем, что то, как операционная система переключается между процессами так же случайно. Может быть как оптимальный случай, когда ОС переключается между процессами в том же порядке, что они и пересылают данные друг другу, так и худший случай, когда переключения между процессами будут в таком порядке, что большая часть процессов будет ожидать какой-либо процесс, переключение к которому будет осуществлено в последнюю очередь. Рост времени выполнения программы с увеличением количества пересылаемых чисел замечен не был, что объясняется высокой эффективностью пересылки данных между процессами.

ПРИЛОЖЕНИЕ А

КОД ПРОГРАММЫ

```
#include <mpi.h>
#include <iostream>
#define GENFLOATNUM 10000

int main(int argc, char** argv) {
    MPI_Init(&argc, &argv);
    int procNum, procRank;
    MPI_Comm_rank(MPI_COMM_WORLD, &procRank);
    MPI_Comm_size(MPI_COMM_WORLD, &procNum);
    int dims=procNum;
    int periods = 1;
    MPI_Comm gridComm;
    MPI_Cart_create(MPI_COMM_WORLD, 1, &dims, &periods, 1, &gridComm);
    float f[GENFLOATNUM];
    float buff[GENFLOATNUM + MPI_BSEND_OVERHEAD];
    MPI_Buffer_attach(buff,      (GENFLOATNUM+      MPI_BSEND_OVERHEAD)      *
sizeof(float));
    for(int i=0;i<GENFLOATNUM;i++)
        f[i] =(float)i+ procRank * 0.01f;
    MPI_Status status;
    MPI_Barrier(gridComm);
    double timer = MPI_Wtime();
    int srcRank, rcvRank;
    MPI_Cart_shift(gridComm, 0, -1, &srcRank, &rcvRank);
    MPI_Bsend(f, GENFLOATNUM, MPI_FLOAT, rcvRank,0, gridComm);
    MPI_Recv(f, GENFLOATNUM, MPI_FLOAT, srcRank,0, gridComm, &status);
    MPI_Barrier(gridComm);
    timer = MPI_Wtime() - timer;
    std::cout << "Proc " << procRank << " recive ";
    for (int i = 0;i < GENFLOATNUM;i++)
        std::cout << f[i] << " ";
    std::cout<<std::endl;
    MPI_Barrier(gridComm);
    if(procRank==0)
        std::cout << "Work time: " << timer * 1000 << std::endl;
    MPI_Comm_free(&gridComm);
    MPI_Finalize();
    return 0;
}
```