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

ОТЧЕТ

по лабораторной работе №1
по дисциплине «Системы параллельной обработки данных»
Тема: Использование функций обмена данными «точка-точка» в библиотеке MPI

Студент гр. 1310 _____ Комаров Д.Е.

Преподаватель _____ Татаринов Ю.С.

Санкт-Петербург
2025

Цель работы

Целью выполнения лабораторной работы является построение программы с использованием функций обмена данными типа «точка-точка» при помощи библиотеки MPI.

Постановка задачи

Вариант 1. Испорченный телефон. Процесс 0 генерирует строковое сообщение и передает его процессу со следующим номером. Процесс-получатель случайным образом меняет в сообщении один символ и передает его дальше. Последний процесс передает получившийся результат «ведущему».

Выполнение работы

Программа, выполняющая поставленную задачу представлена в приложении А. Выполнение программы начинается с определения количества процессов и ранга текущего процесса. В случае, если процесс имеет ранг 0, то он случайно генерирует сообщение длинной MESSAGE_LENGTH и отправляет его процессу с рангом 1. После чего данный процесс ожидает получения итогового сообщения. Если ранг процесса отличен от 0, то он начинает ожидать получения сообщения с рангом, меньшим чем у него на 1. После получения сообщения данный процесс меняет случайный символ в сообщении и отправляет измененное сообщение процессу с рангом, который больше, чем у него, на 1. Процесс с рангом, равным количеству процессов минус 1 отправляет измененное сообщение процессу с рангом 0. После получения итогового сообщения процесс с рангом 0 выводит его в консоль. На рисунке 1 представлен результат работы программы для сообщения длинной 10 и 4 процессов.

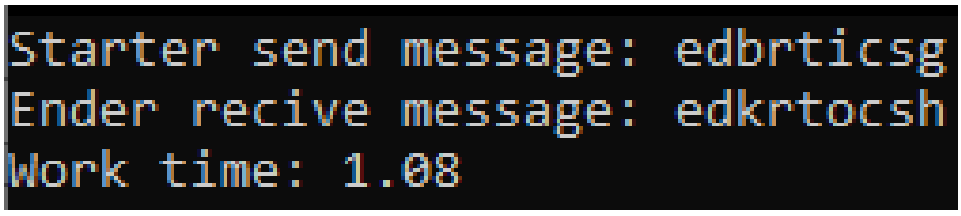

Рисунок 1 – Демонстрация работы программы

На рисунке 2 представлен алгоритм программы в виде сети Петри для четырёх процессов.

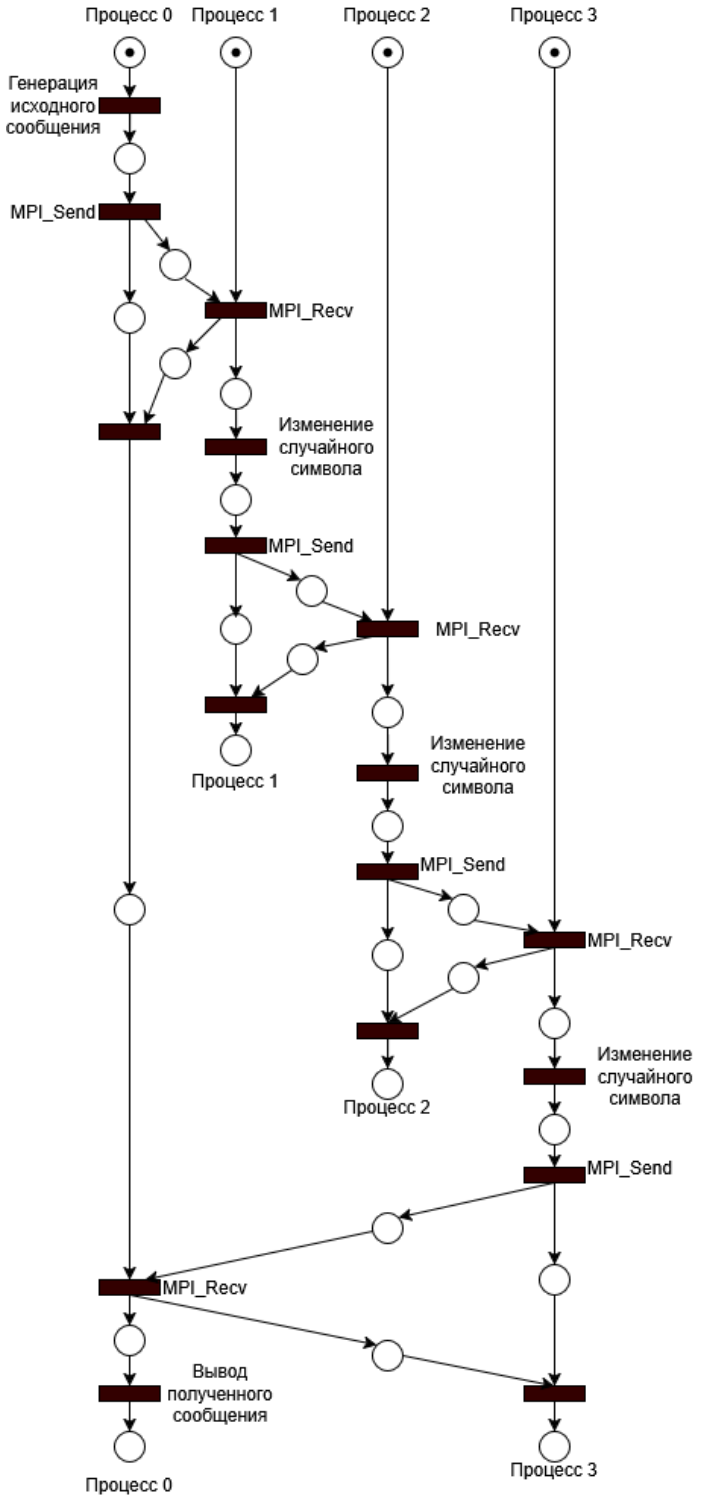

Рисунок 2 – Сеть Петри

Протестируем полученный алгоритм на различном количестве процессов и различной длинны сообщений. В измерения не включено время генерации исходного сообщения и вывода итогового сообщения на экран.

Время работы алгоритма для сообщения длинной 10 символов представлено в таблице 1.

Таблица 1 – Время работы алгоритма для сообщения длинной 10 символов

Число процессов	Время выполнения (мс)
2	0.48
4	1.05
8	2.41
16	4.65
32	9.78

Время работы алгоритма для сообщения длинной 100 символов представлено в таблице 2.

Таблица 2 – Время работы алгоритма для сообщения длинной 100 символов

Число процессов	Время выполнения (мс)
2	0.45
4	1.03
8	2.33
16	4.36
32	9.43

Время работы алгоритма для сообщения длинной 1000 символов представлено в таблице 3.

Таблица 3 – Время работы алгоритма для сообщен длинной 1000 символов

Число процессов	Время выполнения (мс)
2	0.47
4	1.11
8	2.37
16	4.42
32	9.32

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

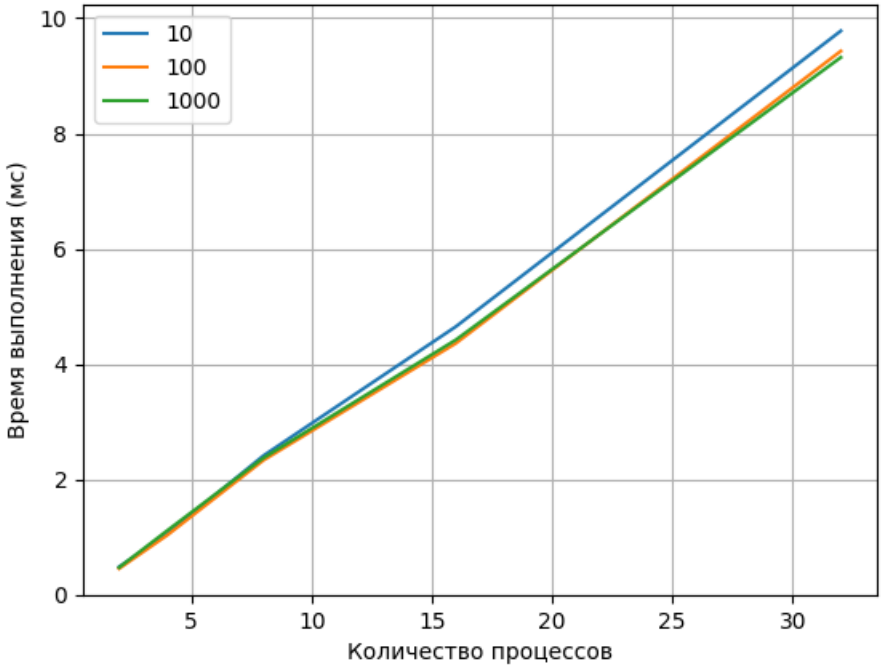

Рисунок 3 – График, отражающий зависимость времени выполнения от количества процессов

На рисунке 4 представлен график, отражающий зависимость времени выполнения от длинны сообщения.

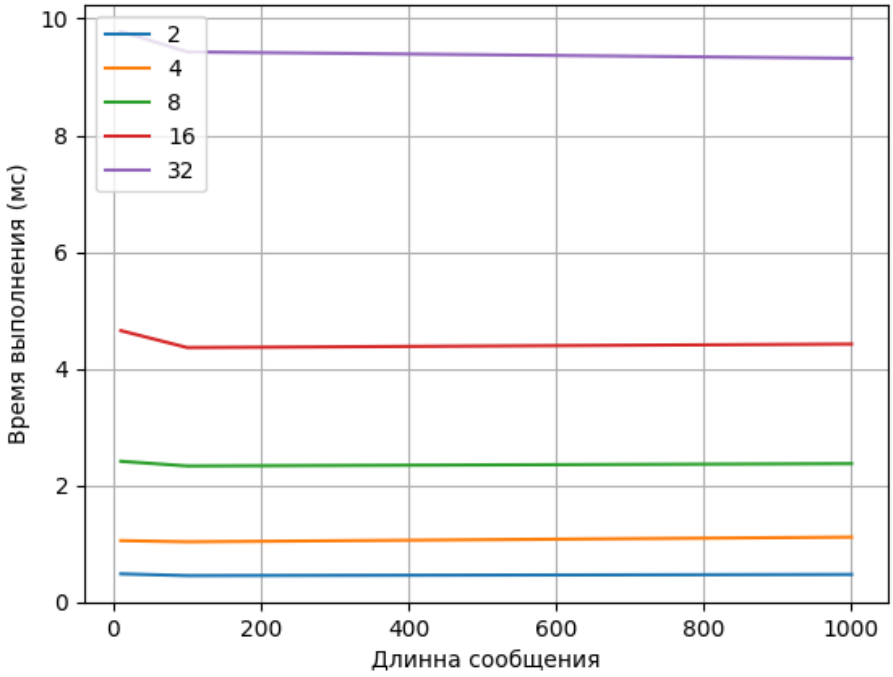

Рисунок 4 – График, отражающий зависимость времени выполнения от длинны сообщения

Из таблиц 1–3 и рисунков 3 и 4 видно, что время выполнения линейно растет с увеличением количества процессов, что соответствует ожиданиям, так как при росте количества процессов сообщение проходит большую цепочку пересылок. Различия во времени для разной длинны сообщения оказались в рамках погрешности, что объясняется высокой эффективностью пересылки данных между процессами. На рисунке 5 представлен график замедления выполнения программы в зависимости от количества процессов.

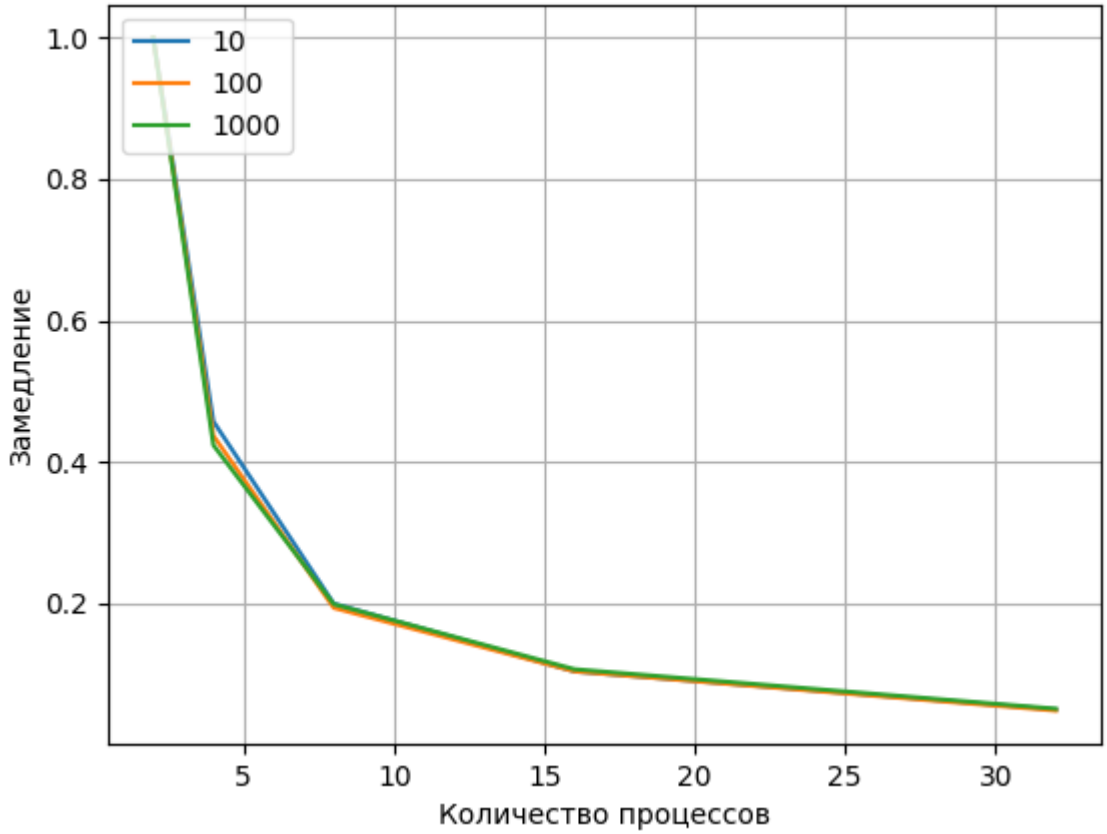

Рисунок 5 – График замедления выполнения программы в зависимости от количества процессов

Выводы

В ходе выполнения лабораторной работы была написана программа с использованием функций обмена данными типа «точка-точка» при помощи библиотеки MPI.

Программы была протестирована для различного количества процессов и различного объема входных данных. Было выяснено, что время выполнения линейно зависит для данной программы от количества процессов, что объясняется тем фактом, что при большем количестве

процессов сообщение проходит большую цепочку передач, пока не достигнет исходного пункта. Также было выяснено, что различия во времени выполнения для различной длинны сообщения находятся в рамках погрешности измерений, что объясняется высокой эффективностью пересылки сообщения между процессами.

ПРИЛОЖЕНИЕ А

КОД ПРОГРАММЫ

```
#include <mpi.h>
#include <iostream>
#include <random>
#include <iomanip>

#define MESSAGE_LENGTH 10

int main(int argc, char** argv) {
    MPI_Init(&argc, &argv);
    int procNum, procRank;
    char message[MESSAGE_LENGTH];
    MPI_Status status;
    MPI_Comm_size(MPI_COMM_WORLD, &procNum);
    MPI_Comm_rank(MPI_COMM_WORLD, &procRank);
    if (procNum < 2) {
        std::cout << "Need 2 or more process" << std::endl;
        return -1;
    }
    srand((int)time(nullptr) + procRank);
    if (procRank == 0) {
        for (int i = 0;i < sizeof(message) - 1;i++)
            message[i] = (rand() % ('z' - 'a') + 'a');
        message[sizeof(message) - 1] = '\0';
        std::cout << "Starter send message: " << message << std::endl;
        double timer = MPI_Wtime();
        MPI_Send(message,    strlen(message)+1,    MPI_CHAR,    1,    0,
MPI_COMM_WORLD);
        MPI_Recv(message,  sizeof(message),  MPI_CHAR,  procNum  -  1,
MPI_ANY_TAG, MPI_COMM_WORLD, &status);
        timer = MPI_Wtime() - timer;
        timer *= 1000;
        std::cout << "Ender recive message: " << message << std::endl;
        std::cout << std::fixed << std::setprecision(2) << "Work time: "
<< timer << std::endl;
    }else {
        MPI_Recv(message,  sizeof(message),  MPI_CHAR,  procRank  -  1,
MPI_ANY_TAG, MPI_COMM_WORLD, &status);
        message[rand()  %  strlen(message)]  =  (rand()  %  ('z'  -  'a')  +
'a');
        MPI_Send(message,  strlen(message)+1,  MPI_CHAR,  (procRank  ==
(procNum - 1)) ? 0 : (procRank + 1), 0, MPI_COMM_WORLD);
    }
    MPI_Finalize();
    return 0;
}
```