

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

Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ
по лабораторной работе №2
по дисциплине «Системы параллельной обработки данных»
Тема: Коллективные операции

Студент гр. 1310

Комаров Д.Е.

Преподаватель

Татаринов Ю.С.

Санкт-Петербург

2025

Цель работы

Целью выполнения лабораторной работы является построение программы с использованием коллективных операций при помощи библиотеки MPI.

Постановка задачи

Вариант 2. В каждом процессе дан набор из 5 целых чисел. Используя функцию MPI_Gather, переслать эти наборы в главный процесс и вывести их в порядке возрастания рангов переславших их процессов (первым вывести набор чисел, данный в главном процессе).

Выполнение работы

Программа, выполняющая поставленную задачу представлена в приложении А. Выполнение программы начинается с определения количества процессов и ранга текущего процесса. После чего каждый процесс генерирует 5 чисел. Числа в каждом процессе генерируются от 0 до 4 и к ним прибавляется умноженный на 5 ранг процесса. Таким образом, процесс с рангом 0 сгенерирует числа 0,1,2,3,4; процесс с рангом 1 сгенерирует 5,6,7,8,9 и т.д. Затем выполняется коллективная операция MPI_Gather, которая пересыпает сгенерированные числа со всех процессов в процесс с рангом 0. После того, как операция проведена, главный процесс выводит полученные числа в порядке увеличения ранга процесса-отправителя (MPI_Gather автоматический сортирует получаемые данные в принимающем буфере в порядке увеличения ранга процесса-отправителя). На рисунке 1 представлен результат работы программы для 4-х процессов.

```
From rank 0: 0 1 2 3 4
From rank 1: 5 6 7 8 9
From rank 2: 10 11 12 13 14
From rank 3: 15 16 17 18 19
Work time: 0.5235
```

Рисунок 1 – Демонстрация работы программы

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

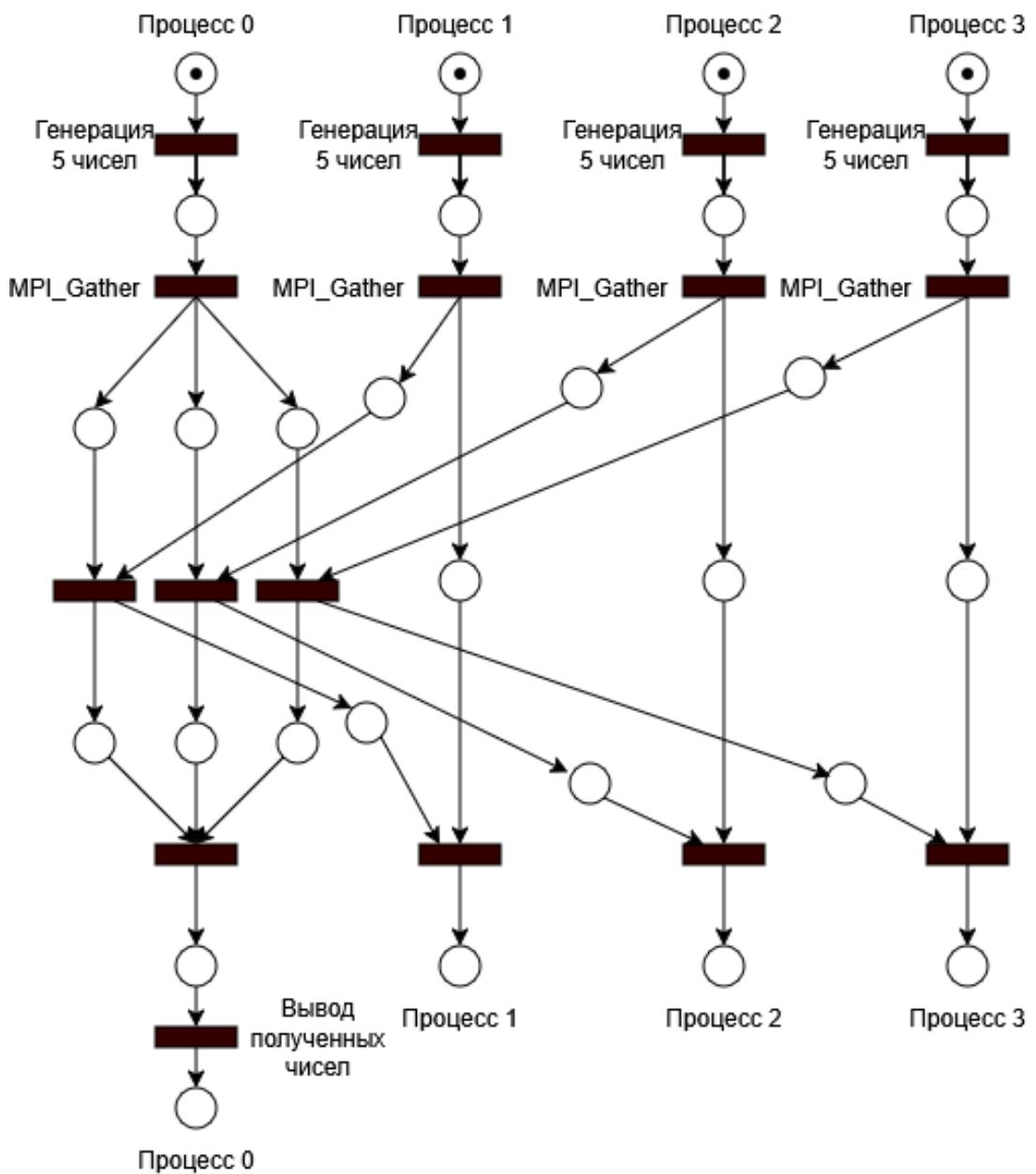


Рисунок 2 – Сеть Петри

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 5 чисел представлено в таблице 1.

Таблица 1 – Время работы алгоритма при пересылке 5 чисел

Число процессов	Время выполнения (мс)
1	0.01

Продолжение таблицы 1

2	0.27
4	0.5
8	1.03
16	2.53
32	5.99

Время работы алгоритма при пересылке 50 чисел представлено в таблице 2.

Таблица 2 – Время работы алгоритма при пересылке 50 чисел

Число процессов	Время выполнения (мс)
1	0.01
2	0.3
4	0.56
8	1.2
16	2.41
32	6.39

Время работы алгоритма при пересылке 5000 чисел представлено в таблице 3.

Таблица 3 – Время работы алгоритма при пересылке 5000 чисел

Число процессов	Время выполнения (мс)
1	0.01
2	0.31
4	0.62
8	1.23
16	2.42
32	6.2

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

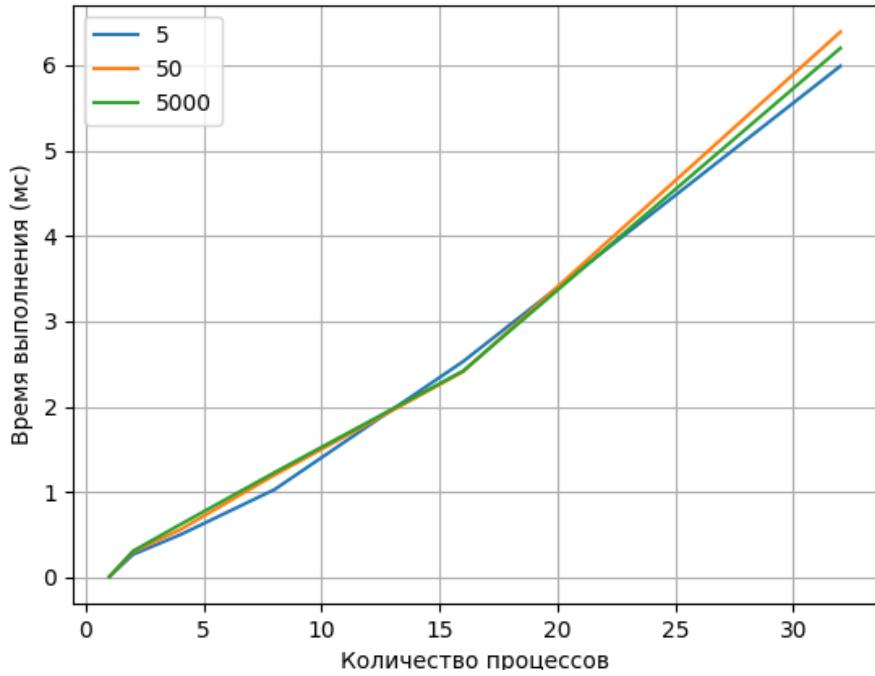


Рисунок 3 – График, отражающий зависимость времени выполнения от количества процессов

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

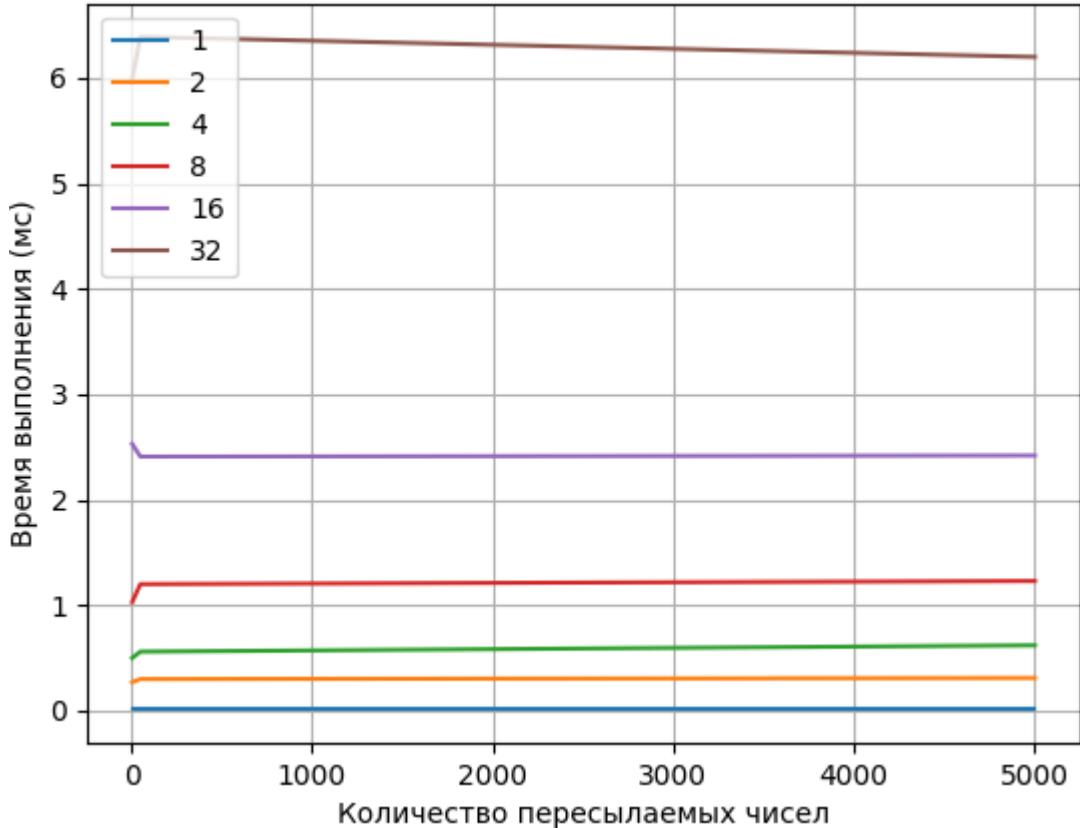


Рисунок 4 – График, отражающий зависимость времени выполнения от количества пересылаемых чисел

Из таблиц 1–3 и рисунков 3 и 4 видно, что время выполнения линейно растет с увеличением количества процессов, что соответствует ожиданиям, так как при росте количества процессов необходимо большее время на то, чтобы собрать с них всех данные. Различия во времени для разного объема пересылаемых данных 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.

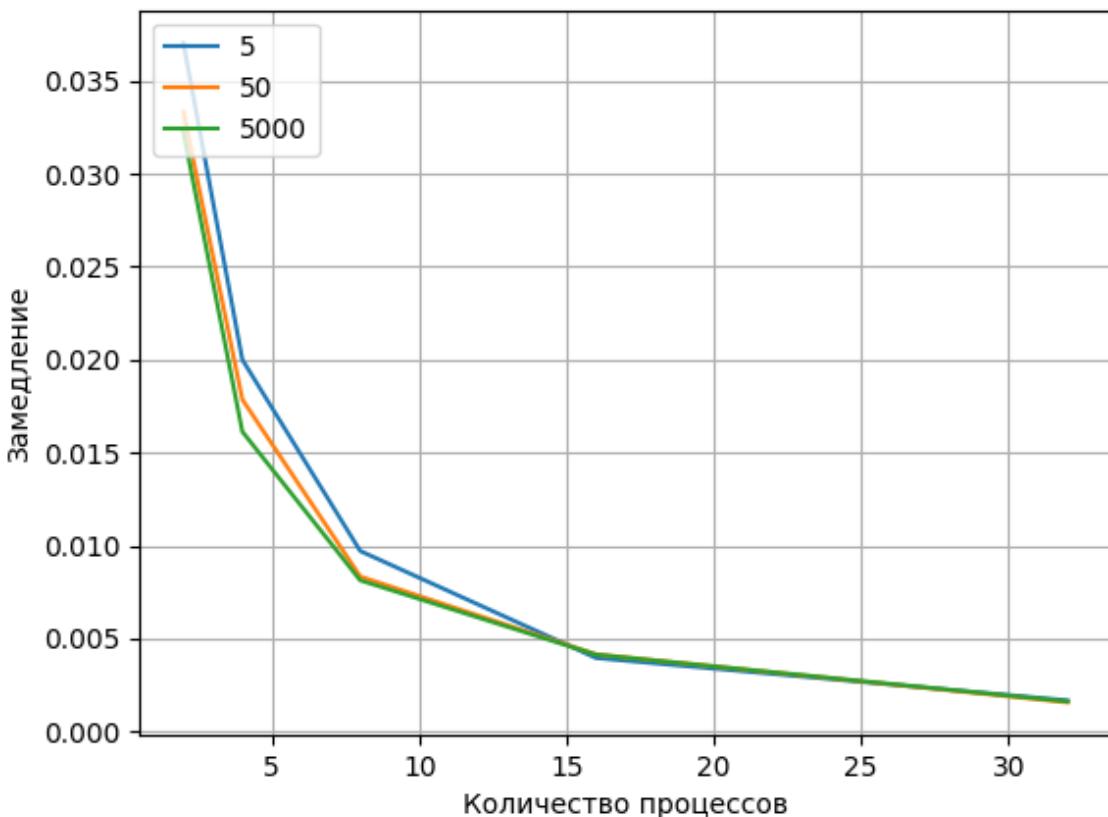


Рисунок 5 – График замедления выполнения программы в зависимости от количества процессов

Выводы

В ходе выполнения лабораторной работы была написана программа с использованием коллективных операций при помощи библиотеки MPI.

Программы были протестированы для различного количества процессов и различного объема данных. Было выяснено, что время выполнения линейно зависит для данной программы от количества процессов, что объясняется тем фактом, что при большем количестве процессов необходимо большее время

на обработку получаемых с них данных. Также было выяснено, что различия во времени выполнения для различного объёма пересылаемых данных находятся в рамках погрешности измерений, что объясняется высокой эффективностью пересылки данных между процессами.

ПРИЛОЖЕНИЕ А

КОД ПРОГРАММЫ

```
#include <mpi.h>
#include <iostream>
#define GENINTNUM 5

int main(int argc, char** argv) {
    MPI_Init(&argc, &argv);
    int procNum, procRank;
    MPI_Comm_size(MPI_COMM_WORLD, &procNum);
    MPI_Comm_rank(MPI_COMM_WORLD, &procRank);
    int arr[GENINTNUM];
    int *rcvarr = new int[procNum * GENINTNUM];
    for (int i = 0; i < GENINTNUM; i++)
        arr[i] = procRank*GENINTNUM+i;
    MPI_Barrier(MPI_COMM_WORLD);
    double timer = MPI_Wtime();
    MPI_Gather(arr, GENINTNUM, MPI_INT, rcvarr, GENINTNUM, MPI_INT, 0,
    MPI_COMM_WORLD);
    timer = MPI_Wtime() - timer;
    if (procRank == 0) {
        for (int i = 0; i < procNum * GENINTNUM; i++) {
            if (i % GENINTNUM == 0)
                std::cout << "From rank "<<i/GENINTNUM<<" : ";
            std::cout << rcvarr[i] << " ";
            if ((i+1)% GENINTNUM ==0)
                std::cout << std::endl;
        }
        std::cout << "Work time: " << timer * 1000 << std::endl;
    }
    delete[] rcvarr;
    MPI_Finalize();
    return 0;
}
```