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

ОТЧЕТ

по лабораторной работе №3
по дисциплине «Системы параллельной обработки данных»
Тема: Группы процессов и коммуникаторы

Студент гр. 1310 _____ Комаров Д.Е.

Преподаватель _____ Татаринов Ю.С.

Санкт-Петербург
2025

Цель работы

Целью выполнения лабораторной работы является построение программы с использованием групп процессов и коммуникаторов при помощи библиотеки MPI.

Постановка задачи

Вариант 3. В каждом процессе, ранг которого делится на 3 (включая главный процесс), даны три целых числа. С помощью функции MPI_Comm_split создать новый коммуникатор, включающий процессы, ранг которых делится на 3. Используя одну коллективную операцию пересылки данных для созданного коммуникатора, переслать исходные числа в главный процесс и вывести эти числа в порядке возрастания рангов переславших их процессов (включая числа, полученные из главного процесса).

Указание. При вызове функции MPI_Comm_split в процессах, которые не требуется включать в новый коммуникатор, в качестве параметра color следует указывать константу MPI_UNDEFINED.

Выполнение работы

Программа, выполняющая поставленную задачу представлена в приложении А. Выполнение программы начинается с определения ранга текущего процесса. После чего при помощи MPI_Comm_split создает новую группу процессов, ранг которых делится на 3. Во всех процессах новой группы создаются 3 целых числа. Числа генерируются от 1 до 3, умножаются на 100 и суммируются с рангом процесса, их сгенерировавшего. Так, процесс с рангом 0 сгенерирует числа 100, 200, 300; процесс с рангом 3 сгенерирует числа 103, 203, 303 и т.д. После чего, при помощи коллективной операции MPI_Gather и нового коммуникатора, данные числа пересылаются в процесс с рангом 0 и выводятся на экран. На рисунке 1 представлен результат работы программы для 4-х процессов.

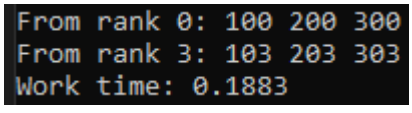

Рисунок 1 – Демонстрация работы программы

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

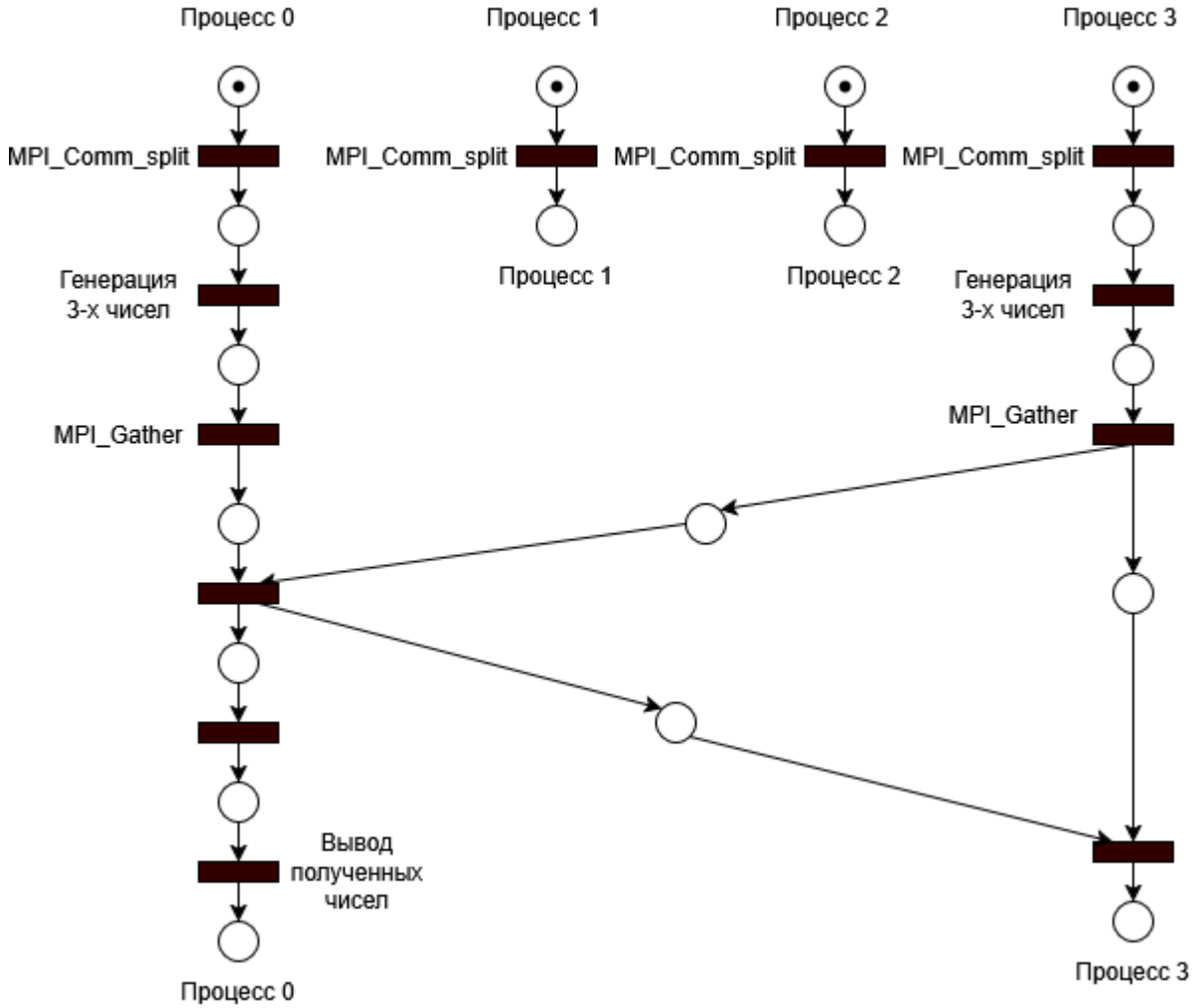

Рисунок 2 – Сеть Петри

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 3 чисел представлено в таблице 1.

Таблица 1 – Время работы алгоритма при пересылке 3 чисел

Число процессов	Время выполнения (мс)
1	0.01
2	0.01
4	0.01

Продолжение таблицы 1

8	0.01
16	0.4
32	2.23

Время работы алгоритма при пересылке 300 чисел представлено в таблице 2.

Таблица 2 – Время работы алгоритма при пересылке 300 чисел

Число процессов	Время выполнения (мс)
1	0.01
2	0.01
4	0.01
8	0.01
16	0.34
32	2.28

Время работы алгоритма при пересылке 3000 чисел представлено в таблице 3.

Таблица 3 – Время работы алгоритма при пересылке 3000 чисел

Число процессов	Время выполнения (мс)
1	0.01
2	0.01
4	0.01
8	0.01
16	0.49
32	2.01

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

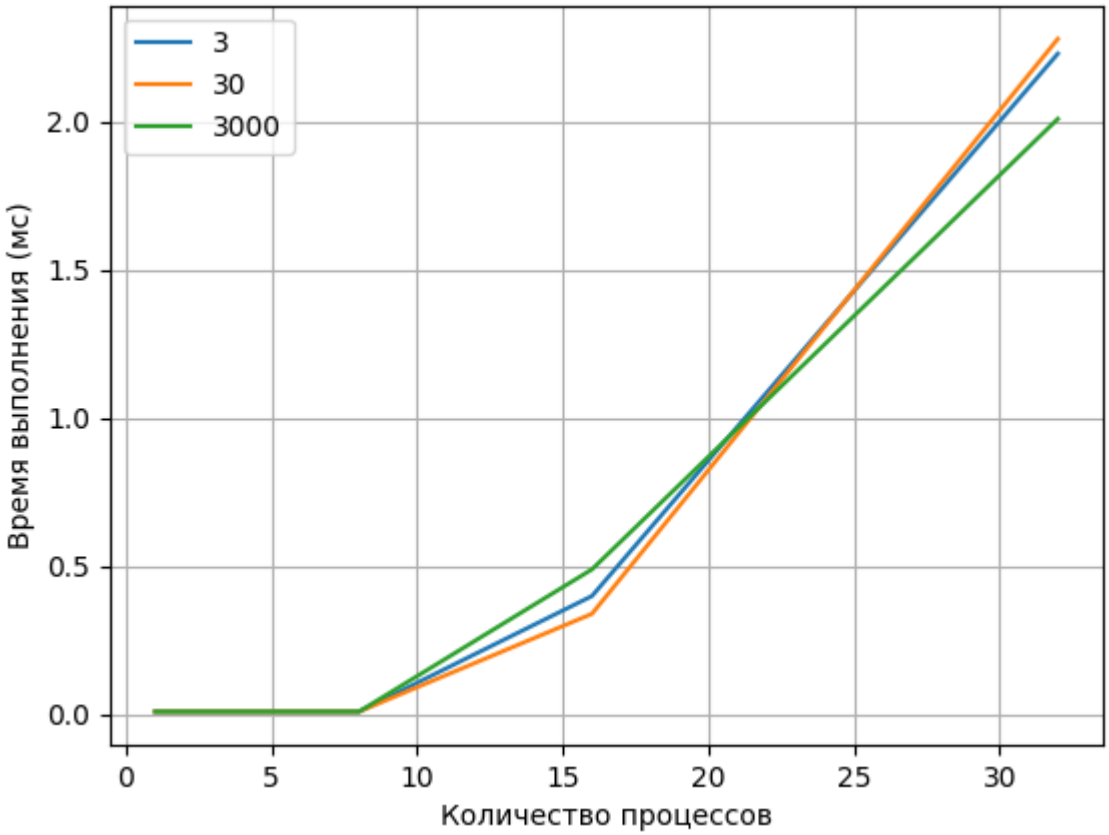

Рисунок 3 – График, отражающий зависимость времени выполнения от количества процессов

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

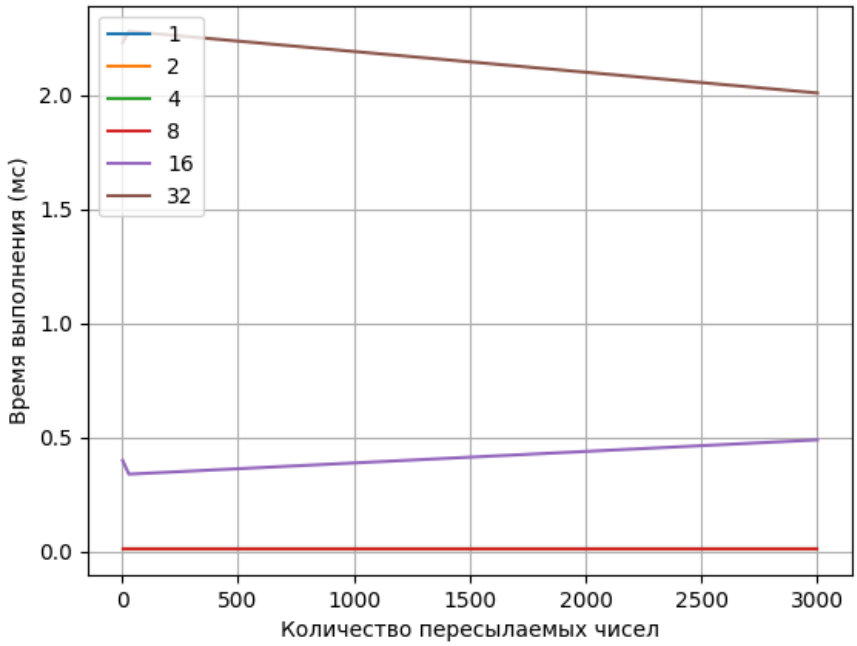

Рисунок 4 – График, отражающий зависимость времени выполнения от количества пересылаемых чисел

Из таблиц 1–3 и рисунков 3 и 4 видно, что время выполнения программы начинает расти после 8 процессов. Объясняется это тем, что, поскольку замеры производились на компьютере с 4-мя ядрами, то для 1, 2, 4, 8 процессов программа выполнялась действительно параллельно (в работе программы всегда участвует только треть процессов из условия, следовательно из 8 процессов, только процессы с рангами 0, 3, 6 активные). Для различного объема данных разницы во времени выполнения замечено не было, что объясняется высокой эффективностью пересылки данных между процессами. На рисунке 5 представлен график замедления выполнения программы в зависимости от количества процессов.

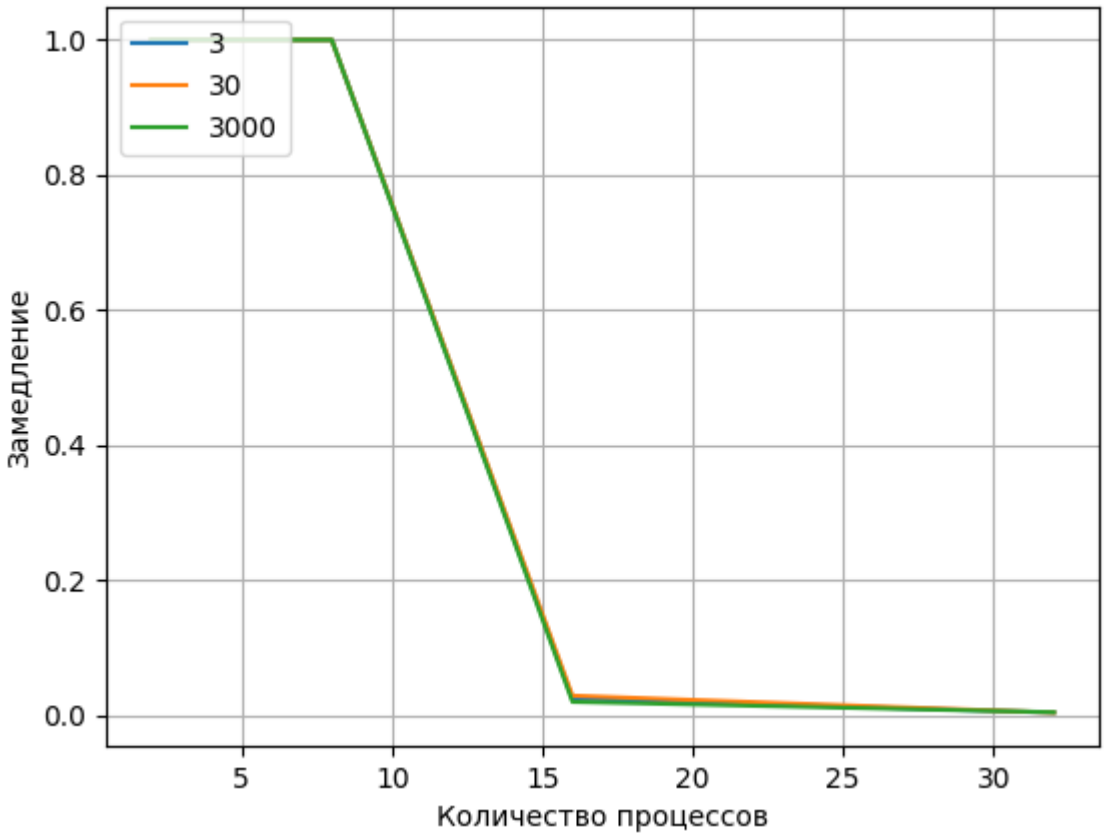

Рисунок 5 – График замедления выполнения программы в зависимости от количества процессов

Выводы

В ходе выполнения лабораторной работы была написана программа с использованием групп процессов и коммуникаторов при помощи библиотеки MPI.

Программа была протестирована для различного количества процессов и различного объема данных. Было выяснено, что время выполнения программы начинает расти после 8 процессов. Объясняется это тем, что, поскольку замеры производились на компьютере с 4-мя ядрами, то для 1, 2, 4, 8 процессов программа выполнялась действительно параллельно (в работе программы всегда участвует только треть процессов, следовательно из 8 процессов, только процессы с рангами 0, 3, 6 активные). Для различного объема данных разницы во времени выполнения замечено не было, что объясняется высокой эффективностью пересылки данных между процессами.

ПРИЛОЖЕНИЕ А

КОД ПРОГРАММЫ

```
#include <mpi.h>
#include <iostream>
#define GENINTNUM 3

int main(int argc, char** argv) {
    MPI_Init(&argc, &argv);
    int procRank;
    MPI_Comm_rank(MPI_COMM_WORLD, &procRank);
    MPI_Comm newComm;
    MPI_Comm_split(MPI_COMM_WORLD, procRank % 3 == 0 ? 1 : MPI_UNDEFINED,
procRank / 3, &newComm);
    if (procRank % 3 == 0) {
        int procNum;
        MPI_Comm_size(newComm, &procNum);
        MPI_Comm_rank(newComm, &procRank);
        int arr[GENINTNUM];
        int *rcvarr = new int[procNum * GENINTNUM];
        for (int i = 0;i < GENINTNUM;i++)
            arr[i] =(i+1)*100+procRank;
        MPI_Barrier(newComm);
        double timer = MPI_Wtime();
        MPI_Gather(arr, GENINTNUM, MPI_INT, rcvarr, GENINTNUM, MPI_INT, 0,
newComm);
        timer = MPI_Wtime() - timer;
        if (procRank == 0) {
            for (int i = 0;i < procNum * GENINTNUM;i++) {
                if (i % GENINTNUM == 0)
                    std::cout << "From rank " << i*3 / 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_Comm_free(&newComm);
    }
    MPI_Finalize();
    return 0;
}
```