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

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Параллельные алгоритмы»

Тема: **Коллективные операции.**

Студент гр. 1384 Усачева Д. В.

Преподаватель Татаринов Ю. С.

Санкт-Петербург

2023

Цель

Изучение коллективных операций в MPI.

Задание

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 приславших их процессов (первым вывести набор чисел, данный в главном процессе).

Выполнение работы

Для выполнения поставленной задачи написана программа на языке C, код которой представлен ниже в листинге 1.

В ходе работы была использована коллективная операция MPI_Gather, опишем ее синтаксис:

int MPI_Gather(void *sbuf, int scount, MPI_Datatype stype, void *rbuf, int rcount, MPI_Datatype rtype, int root, MPI_Comm comm)

- sbuf, scount, stype – параметры передаваемого сообщения,

- rbuf, rcount, rtype – параметры принимаемого сообщения,

- root – ранг процесса, выполняющего сбор данных,

- comm – коммуникатор, в рамках которого выполняется передача данных.

Сначала были инициализированы следующие переменные procNum – общее количество процессов, procRank – ранг текущего процесса, array – массив чисел размерности 5, full_array – массив для сбора информации от других процессов размерности procNum*5.

В цикле для каждого процесса происходит заполнение массива числами от 0 до 5, умноженными на ранг текущего процесса. Далее используется функция MPI_Gather для сбора данных от всех процессов. Нулевой процесс собирает данные в буфере, размещая их в соответствии с рангами процессов-отправителей сообщений, поэтому необходимость в сортировке в порядке возрастания процессов отпадает. Если ранг текущего процесса равен нулю выводятся числа из массивов в порядке возрастания рангов процессов.

Ниже представлена сеть Петри основной части алгоритма (см. рис 1).

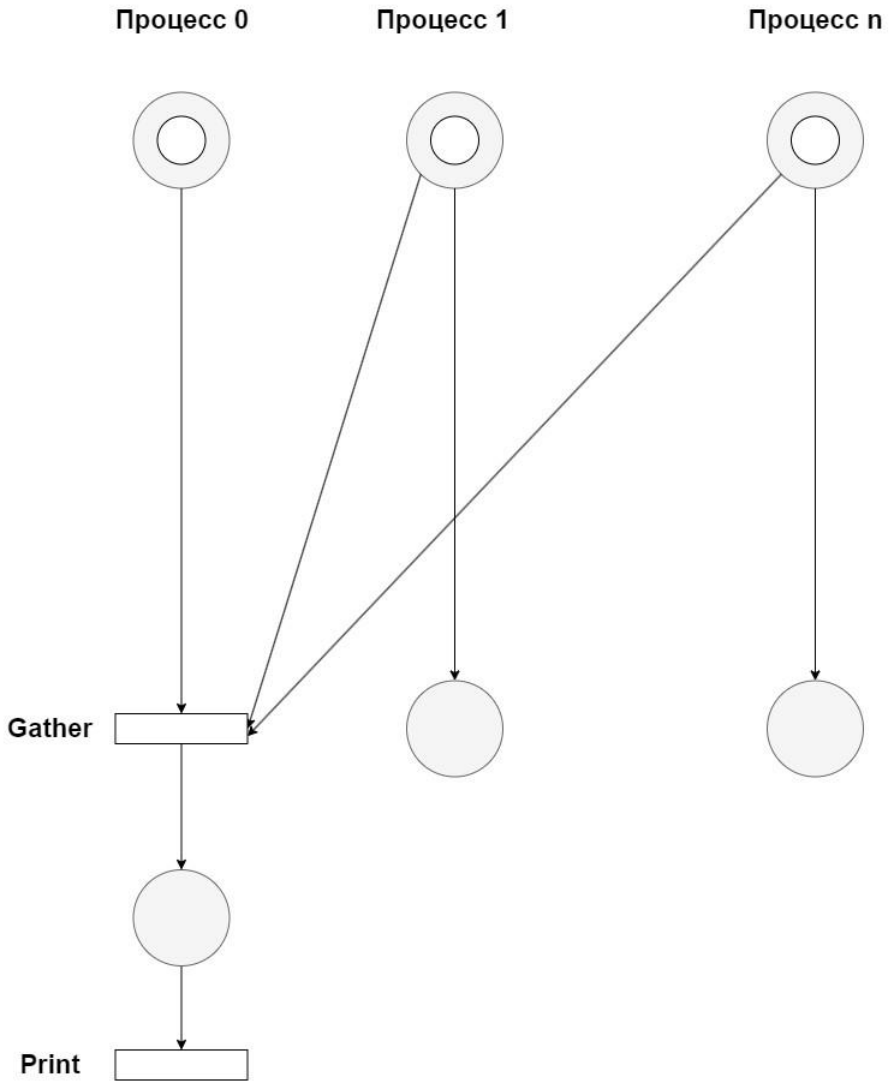

Рисунок 1 — Сеть Петри основной части алгоритма

Листинг 1 — Код программы lab3.c

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

int main(int argc, char** argv) {
    int procNum, procRank;
    int array[5];
    double start;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &procRank);
    MPI_Comm_size(MPI_COMM_WORLD, &procNum);
    int full_array[5 * procNum];
    start = MPI_Wtime();
```

```
        for(int i = 0; i < 5; i++){
            array[i] = procRank * i;
        }
        MPI_Gather(array, 5, MPI_INT, full_array, 5, MPI_INT, 0,
MPI_COMM_WORLD);
        if (procRank == 0) {
            printf("Числа в порядке возрастания рангов процессов:\n");
            for (int i = 0; i < 5 * procNum; i++) {
                printf("%d ", full_array[i]);
                if (i % 5 == 4 ){
                    printf("\n");
                }
            }
            printf("Время работы программы: %f\n", MPI_Wtime() -
start);
        }
        MPI_Finalize();

        return 0;
    }
```

Ниже представлен вывод программы lab3.c

Листинг 2 — Вывод программы lab3.c

```
Числа в порядке возрастания рангов процессов:
0 0 0 0 0
0 1 2 3 4
0 2 4 6 8
0 3 6 9 12
0 4 8 12 16
0 5 10 15 20
0 6 12 18 24
0 7 14 21 28
0 8 16 24 32
0 9 18 27 36
0 10 20 30 40
0 11 22 33 44
0 12 24 36 48
0 13 26 39 52
0 14 28 42 56
```

```
0 15 30 45 60
Время работы программы: 0.000201
```

Так как в условии размер массива фиксированный, рассмотрим зависимость времени работы программы от количества процессов.

Таблица 1 — Среднее время выполнения.

Количество процессов	Среднее время на выполнение(мс)
1	0.042
2	0.061
4	0.080
8	0.098
16	0.184

Ниже указаны графики зависимостей времени выполнения и ускорения (см. рисунки 2-3).

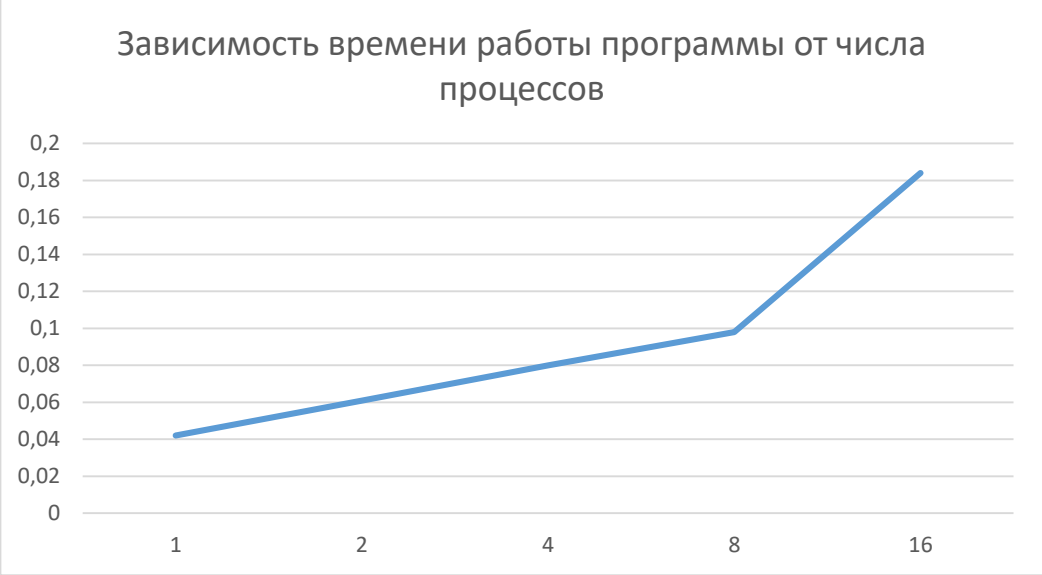

Рисунок 2 — График зависимости времени выполнения от числа процессов

Ускорение времени работы программы можно вычислить по формуле:

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

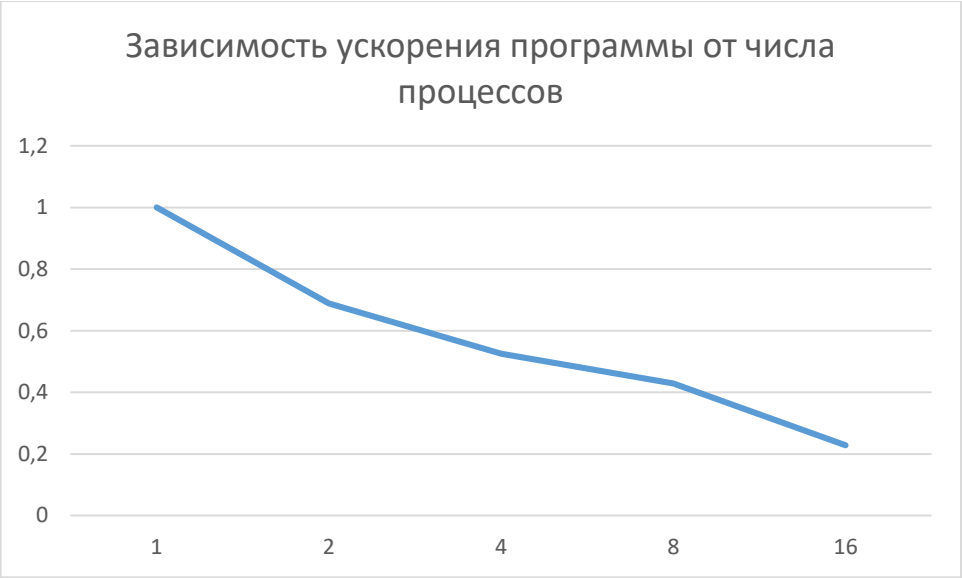

Рисунок 3 — График зависимости ускорения от числа процессов

Выводы

В ходе выполнения лабораторной работы была изучена и использована коллективная операция MPI_Gather.

После полученных экспериментальных результатов можно сделать вывод о том, что время выполнения увеличивается при увеличении числа процессов. Это связано с тем, что нулевому процессу необходимо принять данные от множества других процессов, значит увеличится время ожидания другими процессами своей очереди. Так же увеличивается размер результирующего массива, обрабатываемого нулевым процессом, что тоже влияет на время работы программы.