

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

ОТЧЕТ
по лабораторной работе №4
по дисциплине «Параллельные алгоритмы»
ТЕМА: Группы процессов и коммутаторы.
Создание новых коммутаторов.

Студент

Степаненко Д. В.

Преподаватель

Татаринов Ю. С.

Санкт-Петербург

2023 г.

Цель

Ознакомиться с коммутаторами, их управлением операциями в библиотеке MPI. Написать программу с их использованием функции `MPI_Comm_split`.

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 коммутатор, включающий процессы, ранг которых делится на 3. Используя одну коллективную операцию пересылки данных для созданного коммутатора, 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 коммутатор, в качестве параметра `color` следует указывать константу `MPI_UNDEFINED`.

Выполнение работы

Создается новый коммутатор *new_comm*. Его будет использовать новая группа с помощью функции `MPI_Comm_split()`, в которую входят только процессы, ранг которых делится на 3. Затем каждый процесс с рангом, кратным 3, создает массив *send_numbers*, содержащий 3 произвольных числа. Далее, происходит сбор данных из *send_numbers* в массив *received_numbers* в главном процессе с рангом 0 внутри коммутатора *new_comm* с использованием функции `MPI_Gather()`. Важно отметить, что перед завершением программы, мы освобождаем память, занятую коммутатором *new_comm* с помощью функции `MPI_Comm_free()`. После выполнения программы, в главном процессе с рангом 0 будет выведен список

пересланных чисел, упорядоченных по возрастанию рангов процессов, от которых они были получены. Под конец завершается параллельная часть программы и освобождаются ресурсы.

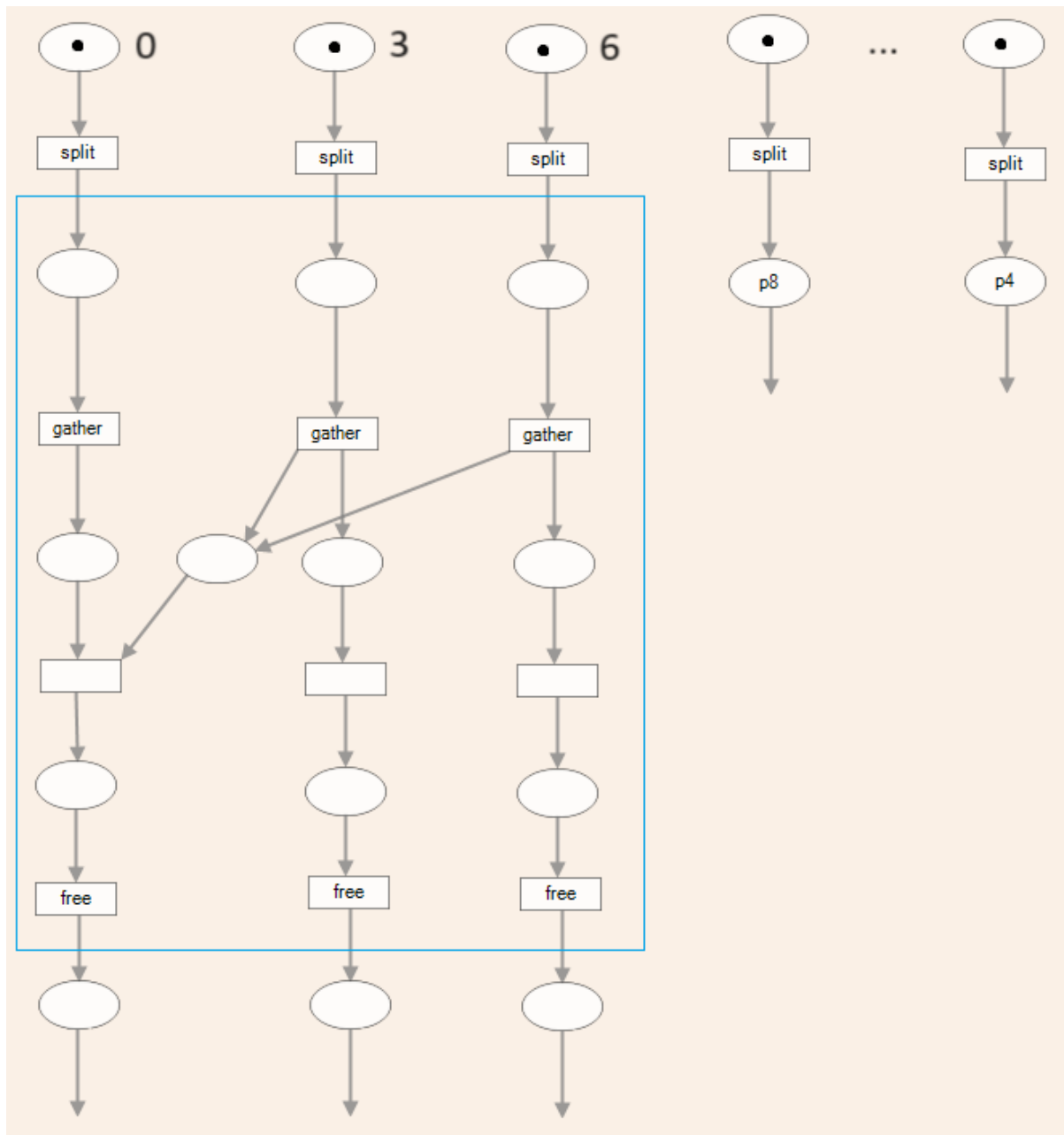


Рисунок 1 - Сеть Петри основной параллельной части программы для семи процессов.

Листинг программы:

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

```

#include <mpi.h>

int main(int argc, char** argv) {
    MPI_Init(&argc, &argv);

    int rank, size;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    // Создаем новый коммуникатор только для процессов, ранг которых
    // делится на 3
    MPI_Comm new_comm;
    MPI_Comm_split(MPI_COMM_WORLD, (rank % 3 == 0) ? 0 :
MPI_UNDEFINED, rank, &new_comm);

    if (rank % 3 == 0) {
        int n = 3; // Количество чисел в каждом процессе
        // Создаем буфер для хранения чисел
        int send_numbers[n];
        for (int i = 0; i < n; i++) {
            send_numbers[i] = rank + i; // Произвольные числа
        }

        // Создаем буфер для хранения пересланных чисел
        int received_numbers[size * n];

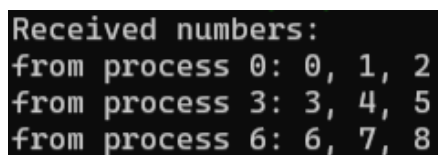
        // Осуществляем пересылку данных в главный процесс
        MPI_Gather(&send_numbers, n, MPI_INT, &received_numbers, n,
MPI_INT, 0, new_comm);

        //освобождаем новый коммуникатор
        MPI_Comm_free(&new_comm);

        // Выводим пересланные числа в порядке возрастания рангов
        // процессов
        if (rank == 0) {
            printf("Received numbers:\n");
            for (int i = 0; i < size; i++){
                for (int j=0; j<size;j+=3){
                    if(received_numbers[i]==j){
                        printf("from process %d: %d, %d, %d\n", j,
received_numbers[i], received_numbers[i+1], received_numbers[i+2]);
                    }
                }
            }
        }

        MPI_Finalize();
        return 0;
    }
}

```



```

Received numbers:
from process 0: 0, 1, 2
from process 3: 3, 4, 5
from process 6: 6, 7, 8

```

Рисунок 2 - Полученный вывод при запуске программы на девяти процессах.

Таблица 1 – Результаты работы программы на разном количестве процессов.

Количество процессов (шт)	Среднее затрачиваемое время (мс)
1	0,073
2	0,061
3	0,094
4	0,139
5	0,1693
6	0,1829
7	0,197
8	0,2021
9	0,2325
10	0,282
11	0,3359
12	0,369

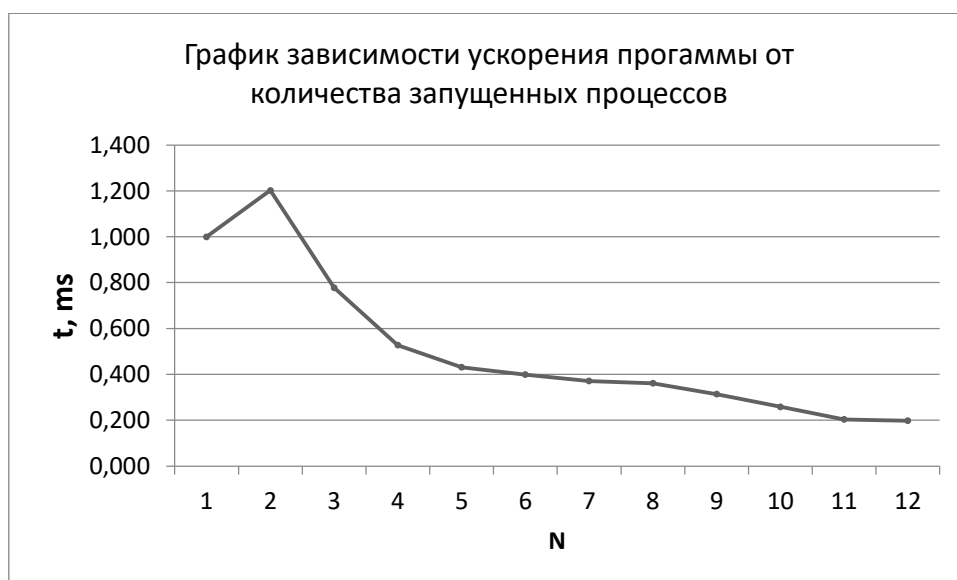


Расчеты ускорения программы выполним по формуле:

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

Количество процессов Р (шт)	Ускорение S_p
1	1
2	1,203
3	0,778
4	0,527
5	0,431
6	0,399
7	0,371
8	0,361
9	0,314
10	0,259
11	0,203
12	0,198

Табл. 2 – Результаты расчетов ускорения программы.



Вывод

В ходе выполнения лабораторной работы были изучены операции с коммутаторами в библиотеке MPI, использована на практике функция *MPI_Comm_split* (). Она расщепляет группу, связанную с родительским коммутатором, на непересекающиеся подгруппы по признаку кратности ранга процесса трем. С ее использованием была написана программа, удовлетворяющая ТЗ.

Время выполнения программы засекалось на параллельной части кода, где использовались функции: *MPI_Comm_split()*, *MPI_Gather()* и *MPI_Comm_free()*. Количество процессов изменялось от 1 до 12, т.к. дальнейшее увеличение было бы малоинформативным. Чем больше процессов, тем больше времени требуется на создание нового коммутатора и пересылку данных в главный процесс. Следовательно, время выполнения будет пропорционально количеству процессов. И чем больше элементов в массиве *send_numbers*, тем больше данных нужно передать. Следовательно, время выполнения будет пропорционально объему данных. Таким образом, время выполнения кода зависит от количества процессов и размера собираемых данных главным процессом.

Исходя из времени выполнения программы, можно сделать вывод, что и ускорение будет уменьшаться с увеличением процессов.