**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 коммуникаторы.**

**Создание новых коммуникаторов.**

|  |  |  |
| --- | --- | --- |
| Студент |  | Степаненко Д. В. |
| Преподаватель |  | Татаринов Ю. С. |

Санкт-Петербург

2023 г.

## Цель

Ознакомиться с коммуникаторами, 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 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

Создается новый коммуникатор *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 коммуникатора *new\_comm* с использованием функции *MPI\_Gather()*. Важно отметить, что перед завершением программы, мы освобождаем память, занятую коммуникатором *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.

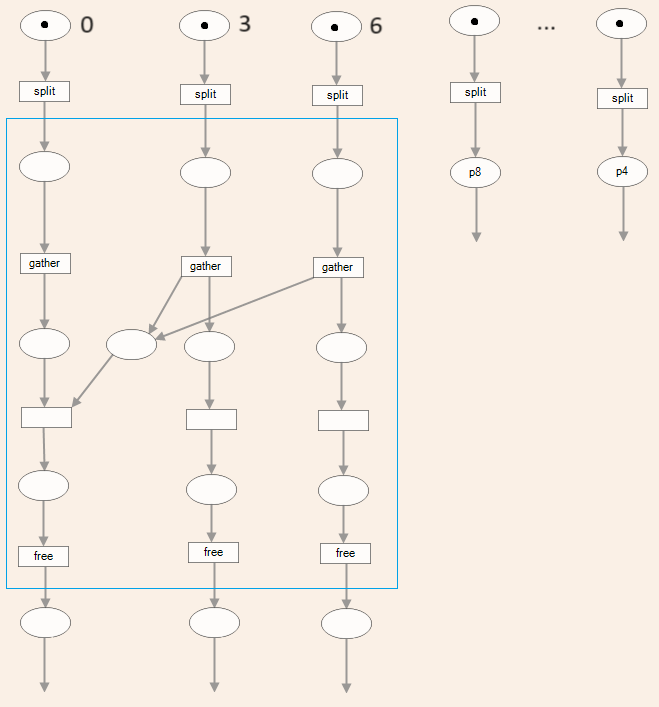


Рисунок 1 - Сеть Петри основной параллельной части программы для семи процессов.

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;

}

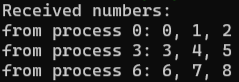


Рисунок 2 - Полученный вывод при запуске программы на девяти процессах.

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:

| Количество процессов P (шт) | Ускорение |
| --- | --- |
| 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 коммуникаторами в библиотеке MPI, использована на практике функция *MPI\_Comm\_split ()*. Она расщепляет группу, связанную с родительским коммуникатором, 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 коммуникатора 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.