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

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Параллельные алгоритмы»

Тема: Коммуникаторы

Студент гр. 8303 _____ Почаев Н.А.

Преподаватель _____ Татаринов Ю.С.

Санкт-Петербург

2020

Задание.

Вариант 9 **-** В каждом процессе дано целое число *N*, которое может принимать два значения: 0 и 1 (имеется хотя бы один процесс с *N=1*). Кроме того, в каждом процессе с *N=1* дано вещественное число *A*. Используя функцию MPI_Comm_split и одну коллективную операцию пересылки данных, переслать числа *A* **во все процессы** с *N=1* и вывести их в порядке возрастания рангов переславших их процессов (включая число, полученное из этого же 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.

Описание решения.

Функция MPI_Comm_split позволяет «расщепить» исходный коммуникатор на набор коммуникаторов, каждый из которых связан с некоторой частью процессов, входящих в исходный коммуникатор. Следует учитывать, что массив новых коммуникаторов в программе не возникает; вместо этого каждому из процессов программы функция MPI_Comm_split предоставляет именно тот коммуникатор из созданного набора, в который входит данный процесс. Предусмотрена также ситуация, когда некоторые процессы не будут включены ни в один из созданных коммуникаторов; для таких процессов функция MPI_Comm_split возвращает «пустой» коммуникатор MPI_COMM_NULL.

Для разбиения процессов на новые группы в функции MPI_Comm_split используется параметр color («цвет»). Все процессы одного цвета включаются в один и тот же новый коммуникатор; при этом любой цвет представляет собой обычное целое число. Предусмотрен также «неопределенный цвет» MPI_UNDEFINED; его надо указывать для процесса, который не следует включать ни в один из новых коммуникаторов. Второй характеристикой, используемой в функции MPI_Comm_split при создании нового набора коммуникаторов, является параметр key («ключ»). Он определяет порядок, в котором будут располагаться процессы в каждом из новых коммуникаторов: процессы в каждом коммуникаторе упорядочиваются *по возрастанию их*

ключей (если некоторые процессы имеют одинаковые ключи, то их порядок определяется средой MPI, которая управляет параллельной программой). Для сохранения в каждом из вновь созданных коммуникаторов исходного порядка следования процессов достаточно в качестве параметра key для каждого процесса указать ранг этого процесса в исходном коммуникаторе.

Для генерации чисел *N* и *A* в данной программе используется генератор случайных чисел из стандартной библиотеки C++.

Результат работы программы представлен на рисунке ниже:

```
❯ ./run.bash
### Process 0 has n = 1###
### Process 1 has n = 0###
### Process 3 has n = 1###
### Process 2 has n = 1###
!!! Process 0 with n = 1:
 from 0 process: 101
 from 2 process: 854
 from 3 process: 46

!!! Process 3 with n = 1:
 from 0 process: 101
 from 2 process: 854
 from 3 process: 46

!!! Process 2 with n = 1:
 from 0 process: 101
 from 2 process: 854
 from 3 process: 46
```

Выводы.

В результате выполнения данной лабораторной работы была изучена и применена на практике работа с коммуникаторами библиотеки MPI.

ПРИЛОЖЕНИЕ А.

ИСХОДНЫЙ КОД ПРОГРАММЫ.

```
#include "mpi.h"
#include <iostream>
#include <cctype>
#include <random>
#include <vector>

// for random
using u32 = uint_least32_t;
using engine = std::mt19937;

int main(int ac,char **av)
{
    MPI_Init(&ac, &av);

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

    if (size == 0) {
          std::cout << "Program has no sence with 0 processes" <<
std::endl;
        return 0;
    }

    // random generation
    std::random_device os_seed;
    const u32 seed = os_seed();
    engine generator(seed);
    std::uniform_int_distribution< u32 > distribute(0, 1);

     MPI_Comm nc;
     int n;
     n = distribute(generator);
     std::cout << "### Process " << rank << " has n = " << n <<
"###" << std::endl;

    // the creation of a new communicator
     MPI_Comm_split(MPI_COMM_WORLD, n, rank, &nc);

     if (n) {
          int a;
        std::uniform_int_distribution< u32 > distribute(1, 1000);
          a = distribute(generator);

        int box[2];
        box[0] = rank;
        box[1] = a;
```

```
          int nc_size;

          // polling the number of processes in the communication
area
          MPI_Comm_size(nc, &nc_size);
          std::vector<int> res(nc_size * 2);

          MPI_Allgather(&box[0], 2, MPI_INT, &res[0], 2, MPI_INT,
nc);

         std::cout << "!!! Process " << rank << " with n = " << n
<< ":" << std::endl;
          for (int i = 0; i < res.size(); ++i) {
               if (i % 2 == 0) {
                 std::cout << " from " << res[i] << " process: ";
            } else {
                 std::cout << res[i] << std::endl;
            }
          }
        std::cout << std::endl;
     }

    MPI_Finalize();

    return 0;
}
```