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. 8303 _____ Почаев Н.А.

Преподаватель _____ Татаринов Ю.С.

Санкт-Петербург

2020

Задание.

Вариант 4 **-** В каждом процессе даны четыре целых числа. Используя функцию MPI_Allgather, переслать эти числа во все процессы и вывести их в каждом процессе в порядке возрастания рангов переславших их процессов (включая числа, полученные из этого же процесса).

Описание решения.

Семейство функций сбора блоков данных от всех процессов группы состоит из четырех подпрограмм: MPI_Gather, MPI_Allgather, MPI_Gatherv, MPI_Allgatherv. Каждая из указанных подпрограмм расширяет функциональные возможности предыдущих.

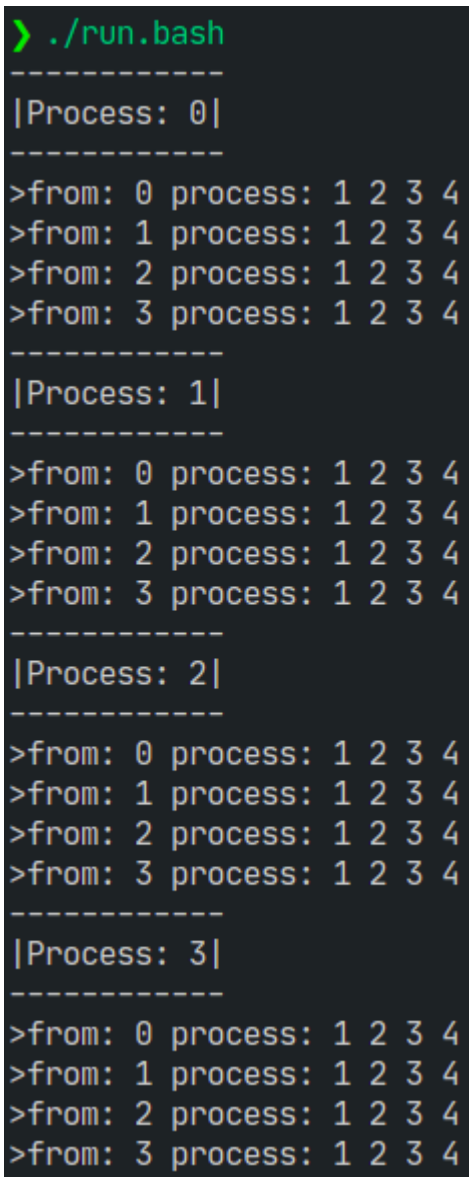
Функция **MPI_Gather** производит сборку блоков данных, посылаемых всеми процессами группы, в один массив процесса с номером root. Длина блоков предполагается одинаковой. Объединение происходит *в порядке увеличения номеров процессов-отправителей*. То есть данные, посланные процессом i из своего буфера sendbuf, помещаются в i-ю порцию буфера recvbuf процесса root. Длина массива, в который собираются данные, должна быть достаточной для их размещения.

Функция **MPI_Allgather** выполняется так же, как MPI_Gather, но получателями являются *все процессы группы*. Данные, посланные процессом i из своего буфера sendbuf, помещаются в i-ю порцию буфера recvbuf каждого процесса. После завершения операции содержимое буферов приема recvbuf у всех процессов одинаково.

```
int MPI_Allgather(void* sendbuf, int sendcount, MPI_Datatype sendtype,void* recvbuf, int recvcount, MPI_Datatype recvtype, MPI_Comm comm)
```

В результате в программе каждый процесс последовательно сначала генерирует массив из 4-х чисел: от 1 до 4, а затем отправляет их коллективной операцией. После этого происходит последовательный вывод полученных данных на экран. Для сохранения номера процесса отправителя используется дополнительная ячейка массива.

Результат работы программы представлен на скриншоте ниже:



Выводы.

В результате выполнения данной лабораторной работы была изучена работа с коллективными операциями библиотеки MPI.

ПРИЛОЖЕНИЕ А.

ИСХОДНЫЙ КОД ПРОГРАММЫ.

```
#include "mpi.h"
#include <vector>

#define SEND_ARR_SIZE 5

int main(int ac,char **av)
{
    int size, rank;

    MPI_Init(&ac, &av);

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

    if (size == 0)
    {
           std::cout << "Mo way to execute with 0 processes" <<
std::endl;
        return 0;
    }

    std::vector<int> n(SEND_ARR_SIZE);
    n[0] = rank;
    for (int i = 1; i < SEND_ARR_SIZE; ++i)
    {
        n[i] = i;
    }

    int t = SEND_ARR_SIZE * size;
    std::vector<int> res(t);

         MPI_Allgather(&n[0],  SEND_ARR_SIZE,  MPI_INT,  &res[0],
SEND_ARR_SIZE, MPI_INT, MPI_COMM_WORLD);

    std::cout << "------------" << std::endl;
    std::cout << "|Process: " << rank << "|" << std::endl;
    std::cout << "------------";
    for (int i = 0; i < res.size(); ++i)
    {
        if (i % SEND_ARR_SIZE == 0) {
               std::cout << std::endl << ">from: " << res[i] << "
process: ";
        } else {
            std::cout << res[i] << " ";
        }
    }
    std::cout << std::endl;
```

```
    MPI_Finalize();

    return 0;
}
```