**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №4**

**по дисциплине «Системы параллельной обработки данных»**

**Тема: «Группы процессов и коммуникаторы. Создание новых коммуникаторов»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 9303 |  | Королёв С.Ю. |
| Преподаватель |  | Татаринов Ю.С. |

Санкт-Петербург

2023

**Цель работы**

Базовое освоение и изучение группировки процессов и создания новых коммуникаторов в библиотеке MPI.

**Формулировка задания**

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 убывания 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.

**Ход работы**

Для изучения коллективных операций в библиотеке MPI был создан скрипт на языке программирования С++ для запуска и обработки результатов запусков процессов.

**Листинг итоговой программы**

#include <iostream>

#include <cctype>

#include <random>

#include <vector>

#include "mpi.h"

using namespace std;

using u32 = uint\_least32\_t;

using engine = mt19937;

int main(int argc, char\*\* argv)

{

MPI\_Init(&argc, &argv);

int rank, size;

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

// random generation

random\_device os\_seed;

const u32 seed = os\_seed();

engine generator(seed);

uniform\_int\_distribution<u32> distribute(0, 1);

int color = distribute(generator);

cout << "Process " << rank << " has n = " << color << endl;

// new communicator

MPI\_Comm nc;

MPI\_Comm\_split(MPI\_COMM\_WORLD, color, rank, &nc);

if (color == 1) {

uniform\_int\_distribution<u32> distribute(1, 1000);

int a = distribute(generator);

int box[2];

box[0] = rank;

box[1] = a;

// polling the number of processes in the communication area

int nc\_size;

MPI\_Comm\_size(nc, &nc\_size);

vector<int> received\_data(nc\_size \* 2);

MPI\_Allgather(&box[0], 2, MPI\_INT, &received\_data[0], 2, MPI\_INT, nc);

for (int i = nc\_size - 1; i >= 0; --i) {

int sender\_rank = received\_data[i \* 2];

int sender\_a = received\_data[i \* 2 + 1];

cout << "Process " << rank << " with n = " << color << " received A from " << sender\_rank << " process: " << sender\_a << endl;

}

}

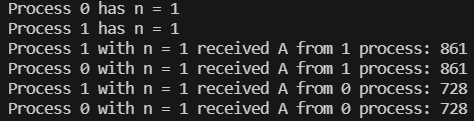
MPI\_Finalize();

return 0;

}

**Результаты работы исходной программы на 2 процессах**

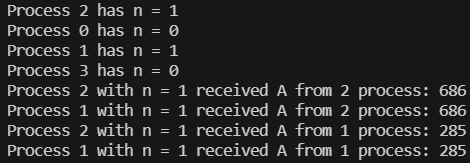
Для 2 процессов:



Для 3 процессов:



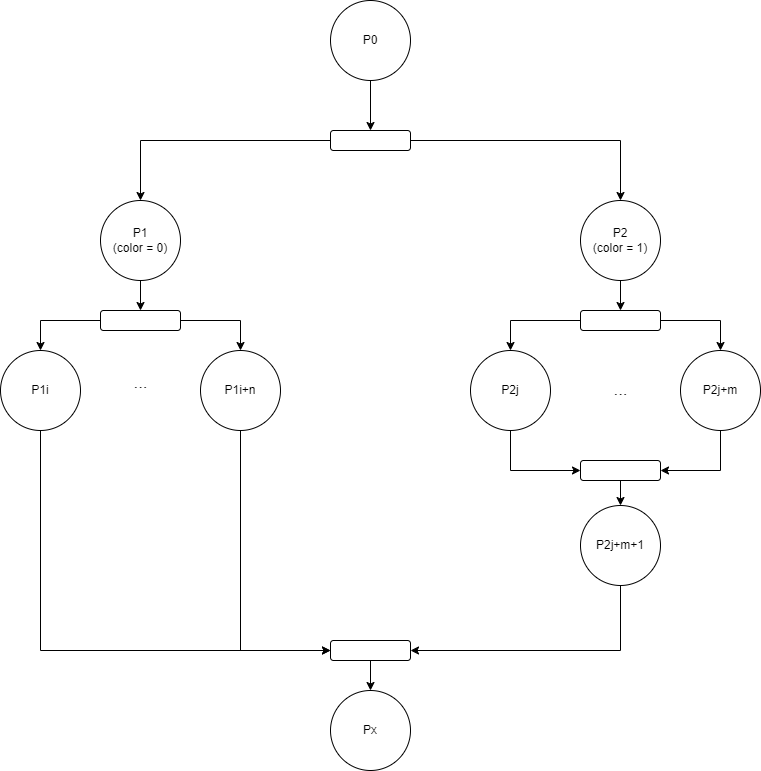
Для 4 процессов:



**Краткое описание выбранного алгоритма решения**

Процессы разбиваются на 2 группы в зависимости от значения случайно выбранного между 0 и 1 параметра color. В процессах с color = 1 случайно генерируется число A в диапазоне от 1 до 1000. Процессы со значением color = 1 отправляются в новый коммуникатор nc с помощью функции MPI\_Comm\_split. В новом коммуникаторе nc собираются их значения А с помощью коллективной пересылки данных через функцию MPI\_Allgather, после чего отображаются с помощью цикла for (в порядке убывания рангов переславших процессов).

**Формальное описание выбранного алгоритма решения с использованием аппарата сетей Петри**



P0 – исходная группа процессов;

P1 – группа процессов с color = 0;

P1i – P1i+n – процессы с color = 0;

P2 – группа процессов с color = 1;

P2j – P2j+m – процессы с color = 1, переход в P2j+m+1 – сбор чисел А c помощью функции MPI\_Allgather.

**График эффективности**

Поскольку значения N для процессов генерируются случайно между 0 и 1, нет зависимости между общим числом процессов и числом группы процессов в коммуникаторе nc (с color = 1), следовательно график эффективности не будет иметь смысла. Однако в ситуации, если количество процессов с color = 1 стабильно росло бы, наблюдался бы рост времени выполнения программы, т.к. увеличение числа процессов обуславливает увеличение числа сообщений и данных для функции MPI\_Allgather.

**Вывод**

В ходе данной работы были изучена и использована функция MPI\_Comm\_split для разбиения процессов на группы с новым коммуникатором. С помощью данной функции было организовано разбиение процессов на две группы, в одной из которых была организована коллективная пересылка данных с помощью функции MPI\_Allgather.