**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:**Группы процессов и коммуникаторы.**

|  |  |  |
| --- | --- | --- |
| Студент гр. 1384 |  | Галенко А.С. |
| Преподаватель |  | Татаринов Ю.С. |

Санкт-Петербург

2023

## Цель работы.

Целью работы является изучение функция библиотеки MPI для управления коммуникаторами и группами процессов, а также разработка программы, демонстрирующей создание нескольких групп процессов, которые выполняют определенную коллективную операцию на уровне свой группы.

## Задание.

***Вариант 10.*** В каждом процессе дано целое число N, которое может принимать два значения: 1 и 2 (имеется хотя бы один процесс с каждым из возможных значений). Кроме того, в каждом процессе дано целое число A. Используя функцию MPI\_Comm\_split и одну коллективную операцию пересылки данных, переслать числа A, данные в процессах с N = 1, во все процессы с N = 1, а числа A, данные в процессах с N = 2, во все процессы с N = 2. Во всех процессах вывести полученные числа в порядке возрастания рангов переславших их процессов (включая число, полученное из этого же процесса).

Указание. Программа должна содержать единственный вызов функции MPI\_Comm\_split, создающий оба требуемых коммуникатора (каждый в соответствующей группе процессов).

## Выполнение работы.

Алгоритм для i-ого процесса:

В завсимости от четности ранга процесса глобального коммуникатора выбирается число N, 2 для четного, 1 для нечетного. Число A выбирается как значение глобального ранга + 1.

Далее с помощью функции MPI\_Comm\_split происходит расщепление исходной группы на две, где в качестве color задается значение N. После расщепления с помощью функций MPI\_Comm\_rank() и MPI\_Comm\_size() процесс получает размер группы и свой ранг в ней.

Затем для всех процессов выделяется память для массива A\_array, который будет содержать все значения A. Его размер равен размеру группы.

С помощью операции MPI\_Allgather() происходит обобщенная передача от всех процессов всем процессам в пределах своей группы значения А.

Далее каждый процесс выводит свой ранг в глобальный и групповой ранги, а после в порядке возрастания рангов переславших значения A процессов печатает массив A\_array.

Формальное описание алгоритма с использованием аппарата Сетей Петри.

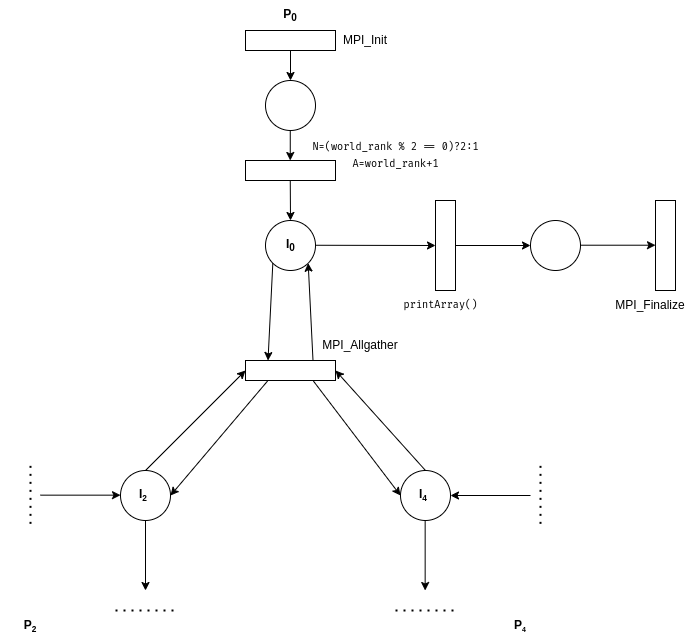


Рисунок 2 – Сеть Петри

## Пример работы программы на 4 процессах представлен на рис.2.

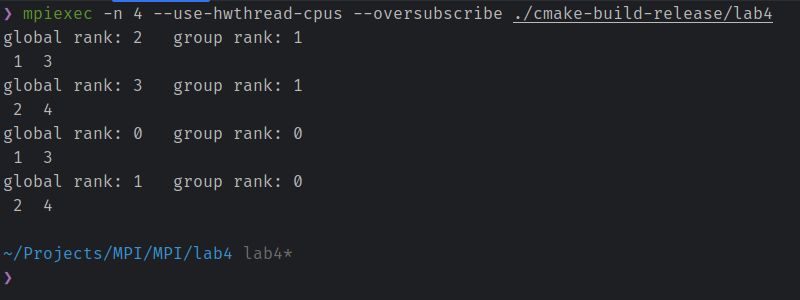


Рисунок 2 – Пример работы программы

## Пример работы программы на 8 процессах на рис.3.

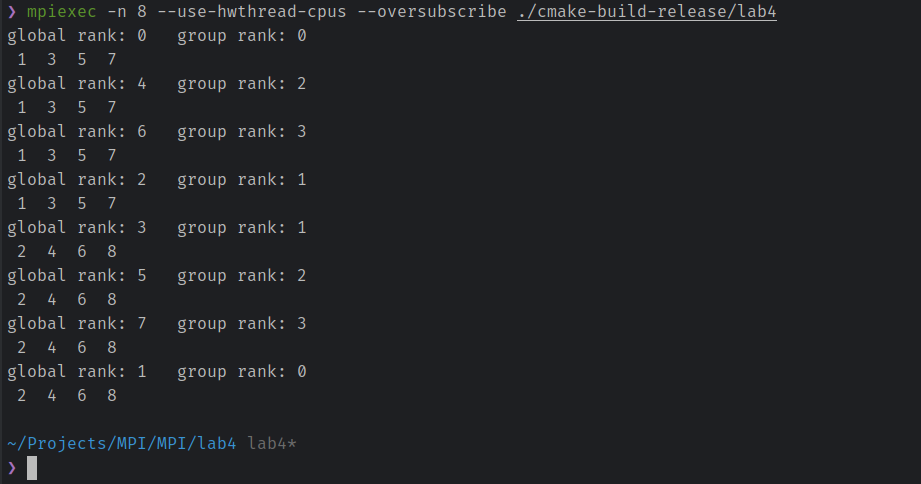


Рисунок 3 – Пример работы программы

## Протестируем работу программы:

Мы будем увеличивать количество, участвующих процессов, чтобы увеличивать размер получаемого для вывода массива, а для измерения времени будем использовать функции MPI\_Barrier() и MPI\_Wtime().

На рис. 4 представлен график зависимости времени выполнения от числа процессов в программе.

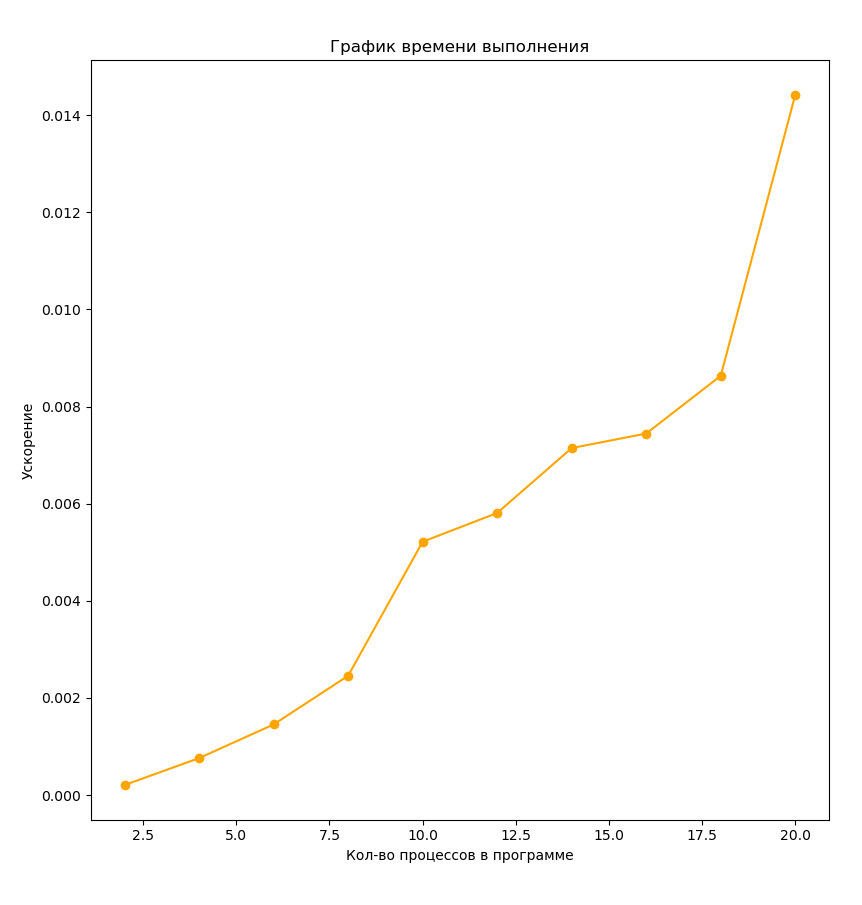


Рисунок 4 – График времени

На рис. 5 представлен график ускорения времени(замедления) выполнения программы от числа процессов для разного объема исходных данных.

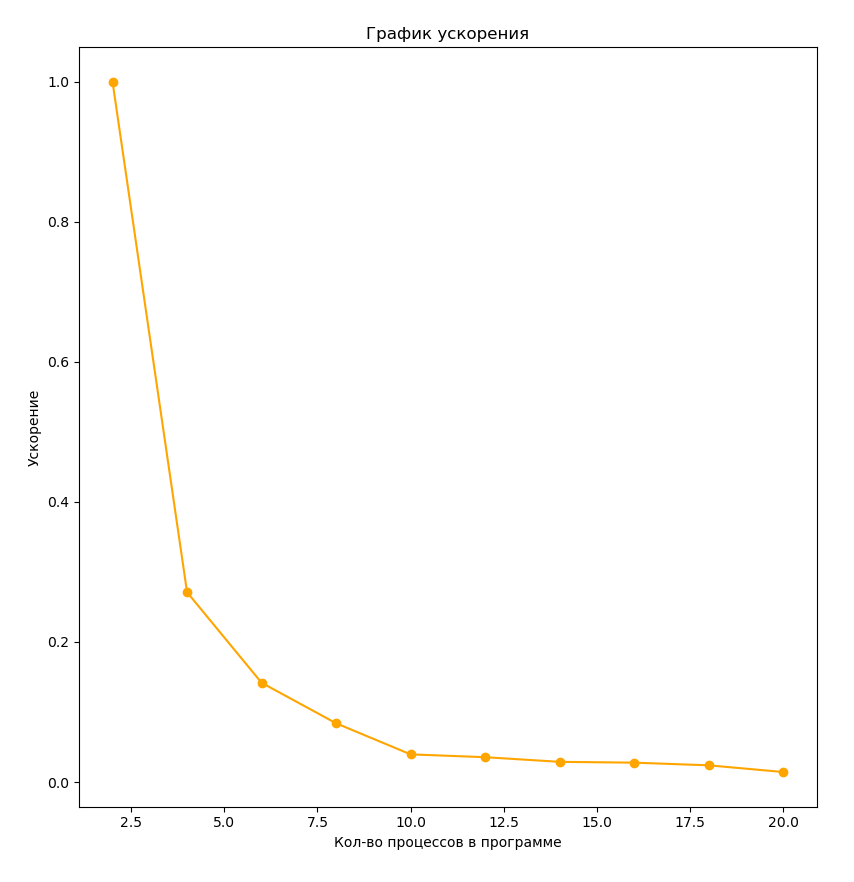


Рисунок 5 – График замедления

Как видно с графиков, с увеличением числа, скорость работы программы существенно замедляется сразу уже после 4 процессов. Это можно объяснить тем, что с увеличение числа процессов растет и число данных, так как каждый процесс содержит одно входное значение А. При этом каждый процесс выводит все значения А. Таким образом оптимизации не происходит в силу того, что добавление процессов увеличивает количество работы программы.

## Выводы.

В ходе данной лабораторной работы были изучены функция библиотеки MPI, которая позволяет расщеплять исходную группу на две непересекающиеся подгруппы, с которыми в дальнейшем модно проводить операции как и с обычной группой. Была написана программа, демонстрирующая работу функции MPI\_Comm\_split, которая производит расщепение, и MPI\_Allgather, которая производит обобщенную передачу от всех процессов всем процессам. С помощью последней каждый процесс напечатал массив значений A своей группы в порядке возрастания рангов процессов. В работе было проведено тестирование по времени выполнения программы, в ходе которого было установлено, что скорость работы программы существенно замедляется после запуска работы программы на 4 процессов и выше. Это происходит, так как с увеличение числа процессов растет и число обрабатываемых данных.

# Приложение А

Название файла: main.cpp

#include <mpi.h>

#include <algorithm>

#include <random>

const int ROOT = 0;

// Вариант 10

void printArray(int len, int \*array);

int main(int argc, char \*argv[]) {

MPI\_Init(&argc, &argv);

int world\_rank, world\_size;

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

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

int N = (world\_rank % 2 == 0) ? 2 : 1;

int A = world\_rank + 1;

MPI\_Barrier(MPI\_COMM\_WORLD);

double t1 = MPI\_Wtime();

MPI\_Comm group\_comm;

MPI\_Comm\_split(MPI\_COMM\_WORLD, N, world\_rank, &group\_comm);

int group\_rank, group\_size;

MPI\_Comm\_rank(group\_comm, &group\_rank);

MPI\_Comm\_size(group\_comm, &group\_size);

int\* A\_array = (int\*) malloc(sizeof (int) \* group\_size);

MPI\_Allgather(&A, 1, MPI\_INT,

A\_array, 1, MPI\_INT, group\_comm);

printf("global rank: %d \t group rank: %d\n",

world\_rank, group\_rank);

printArray(group\_size, A\_array);

MPI\_Comm\_free(&group\_comm);

MPI\_Barrier(MPI\_COMM\_WORLD);

double t2 = MPI\_Wtime();

if(world\_rank == 0)

printf("Time: %f\n", t2 - t1);

MPI\_Finalize();

return 0;

}

void printArray(int len, int \*array) {

for(int i = 0; i < len; i++){

printf(" %d ", array[i]);

if((i + 1) % 12 == 0){

printf("\n");

}

}

printf("\n");

}