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. 3384 _____ Рудаков А.Л.

Преподаватель _____ Татаринов Ю.С.

Санкт-Петербург
2025

Цель работы.

Изучение алгоритмов разделения процессов на группы и создания новых коммуникаторов в MPI и использование их на практике, путем написания параллельных программ, запускаемых на различном числе одновременно работающих процессов

Задание.

Вариант 8.

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.

Выполнение работы.

Для выполнения задания написан код на языке программирования C++, при помощи средств MPI разделяющий программу на параллельную между функциям *MPI_Init(&argc, &argv)* и *MPI_Finalize(),* а также реализующий для каждого процесса определение его ранга, посредством функции *MPI_Comm_rank(MPI_COMM_WORLD, &proc_rang)* и общее количество процессов при помощи функции *MPI_Comm_size(MPI_COMM_WORLD, &num_proc)*.

Создана структура *RangValue* для хранения значения *int rang* и *float value*, для записи туда информации, передаваемой в дальнейшем в сообщениях между

процессами. Для этой структуры создан аналог для сообщений: *MPI_Datatype MPI_RANG_VALUE*.

В начале программы в каждом процессе случайным образом генерируется значение *N* (0 или 1), после чего, если *N* = 1, то генерируется вещественное число от -5 до 5. Далее идет инициализация нового коммуникатора *MPI_Comm new_comm*, а также процессы разделяются на группы (определяются значения color для всех процессов). После этого идет расщепление группы *MPI_Comm_split(..)*, в следствие чего в новый новый коммуникатор начинает следить за созданной подгруппой.

Далее, если процесс находиться в новой группе, то внутри него определяется его новый ранг и общее число процессов в группе, а также инициализируется тип *MPI_RANG_VALUE*. После этого выполняется коллективная операция *MPI_Allgather(..)*, после чего сортировка полученных данных по убыванию ранга процесса, от которого получен результат, за которым следует вывод результата в процессе с новым рангом 0.

Схема Петри представлена на рис.1.

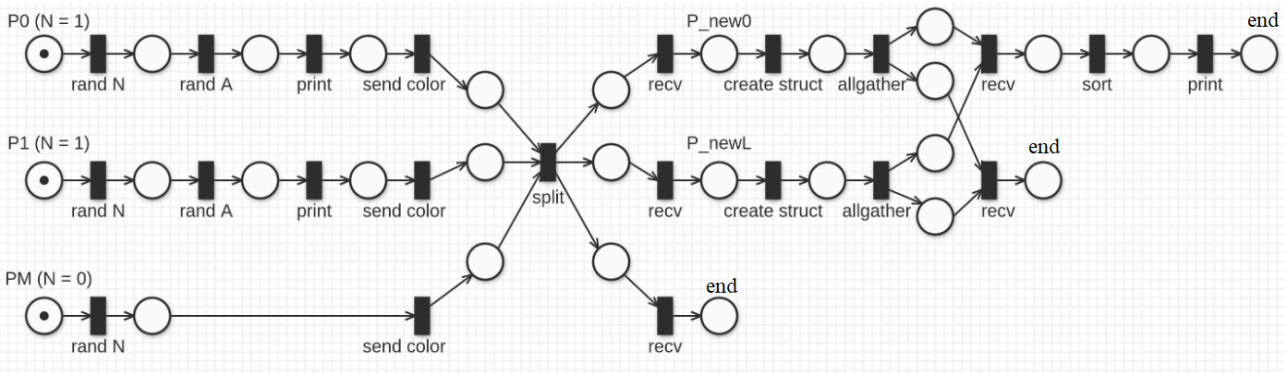

Рисунок 1 — Схема Петри.

Код, написанный для задания 1 на языке программирования C++ см. Приложение А.

Результаты работы программы см. Приложение В.

Зависимость времени работы программы от количества процессов можно увидеть в табл.1 и на рис.2.

Табл.1 см. Приложение Б.

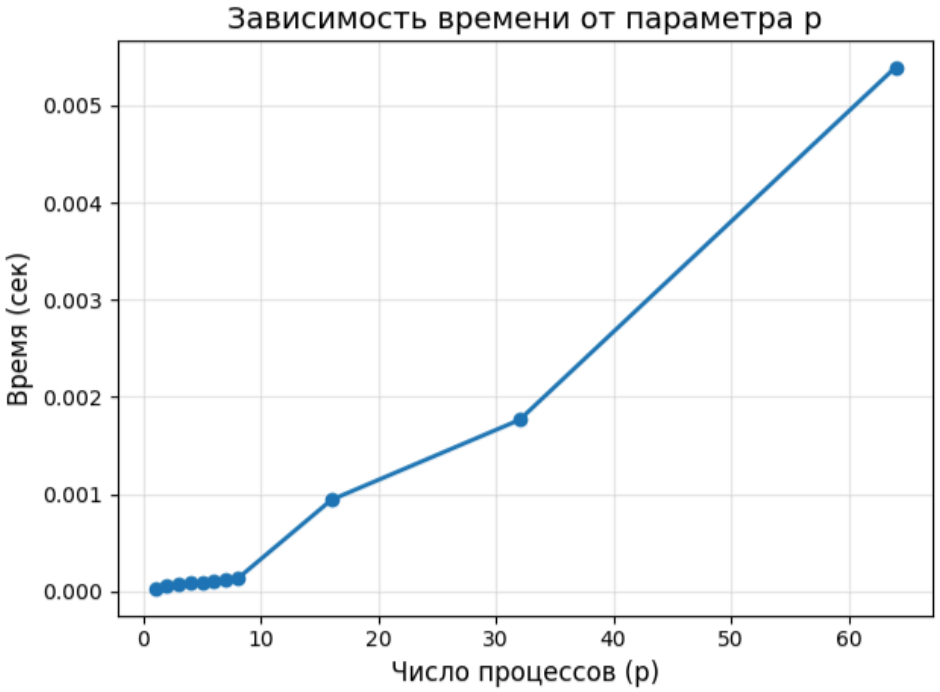

Рисунок 2 - График зависимости времени от числа процессов

На рис.3 изображен график зависимости ускорения от числа процессов.

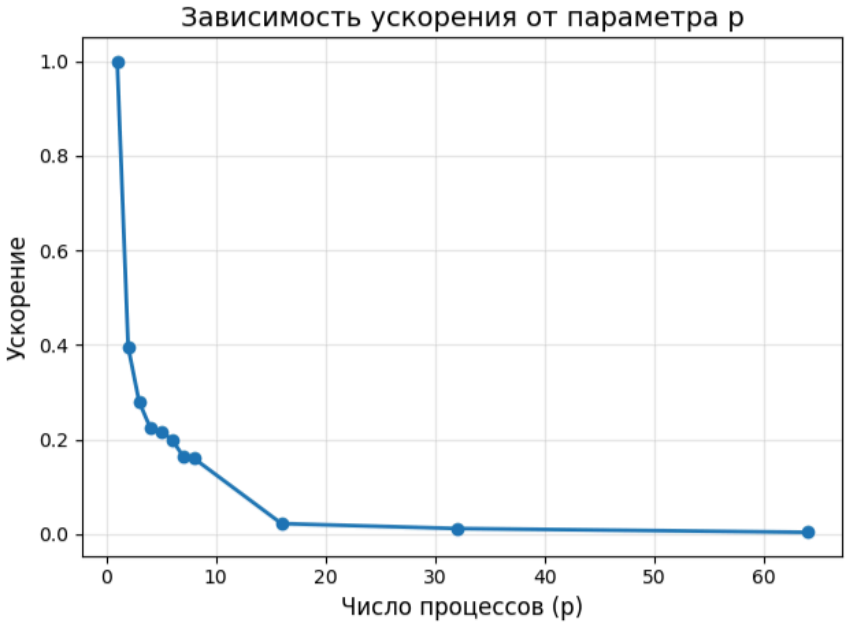

Рисунок 3 - График зависимости ускорения от числа процессов

Из полученных результатов можно заметить, что время растет линейно относительно количества процессов, на которых запускается программа, также можно заметить, что при количестве процессов от 2 до 8 время примерно равно, что обуславливается тем, что количество данных внутри каждый раз увеличивается на одну единицу данных (или две, если N = 1), что составляет O(1) в формате сложности. При этом при бОльших количествах процессов

время увеличивается и ускорение падает. Это обуславливается тем, что на компьютере, на котором проводились замеры, одновременно работают только 8 потоков, вследствие чего при разбиении на большее количество процессов включается планировщик, который распределяет время среди потоков, вследствие чего время увеличивается и ускорение уменьшается.

Выводы.

Изучены алгоритмы разделения процессов на группы и созданы новые коммуникаторы в MPI, в частности использован MPI_Comm_split, на основании которых написана параллельная программа, запускаемая на различном числе одновременно работающих процессов. Проверена работоспособность программы на различном числе процессов, а также замерено время ее работы и ускорение. Выявлено увеличение времени работы на большом количестве запускаемых процессов, что вызвано увеличением накладных расходов на работу планировщика.

ПРИЛОЖЕНИЕ
ИСХОДНЫЙ КОД

Файл mpi_lb4.cpp:

```
#include <iostream>
#include <vector>
#include <mpi.h>
#include <random>
#include <fstream>

struct RangValue {
    int rang;
    float value;
};

int main(int argc, char** argv) {
    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_int_distribution<> distrib_int(0, 1);
    std::uniform_real_distribution<> distrib_real(-5,5);

    MPI_Init(&argc, &argv);

    int proc_rang, num_proc;
    MPI_Comm_rank(MPI_COMM_WORLD, &proc_rang);
    MPI_Comm_size(MPI_COMM_WORLD, &num_proc);

    int N = proc_rang ? distrib_int(gen) : 1;
    float real_value = 0;
    if (N){
        real_value = distrib_real(gen);
        std::cout << "Start proc rang: " << proc_rang << "; value: " << real_value << "\n";
    }

    MPI_Comm new_comm;
    int color = N ? 0 : MPI_UNDEFINED;

    MPI_Comm_split(MPI_COMM_WORLD, color, proc_rang, &new_comm);

    if (N){
        int new_rang, new_size;
        MPI_Comm_rank(new_comm, &new_rang);
```

```
            MPI_Comm_size(new_comm, &new_size);

            MPI_Datatype MPI_RANG_VALUE;
            int blocklengths[2] = {1, 1};
            MPI_Aint displacements[2];
            MPI_Datatype types[2] = {MPI_INT, MPI_FLOAT};

            RangValue temp;
            MPI_Get_address(&temp.rang, &displacements[0]);
            MPI_Get_address(&temp.value, &displacements[1]);
            displacements[1] -= displacements[0];
            displacements[0] = 0;

            MPI_Type_create_struct(2, blocklengths, displacements, types, &MPI_RANG_VALUE);
            MPI_Type_commit(&MPI_RANG_VALUE);

            RangValue sended_data = {new_rang, real_value};
            std::vector<RangValue> all_data(new_size);

            MPI_Allgather(&sended_data, 1, MPI_RANG_VALUE, all_data.data(), 1, MPI_RANG_VALUE,
new_comm);

             if (new_rang == 0){
                 for (int i = 1; i < new_size; i++){
                         int idx = i;
                         while (idx > 0 && all_data[idx].rang > all_data[idx - 1].rang){
                            RangValue temp_data = all_data[idx];
                            all_data[idx] = all_data[idx - 1];
                            all_data[idx - 1] = temp_data;
                            idx--;
                         }
                     }
               for (const auto& data : all_data) {
                   std::cout << "New prog rang " << data.rang << "; value: " << data.value << std::endl;
               }
            }

            MPI_Type_free(&MPI_RANG_VALUE);
            MPI_Comm_free(&new_comm);
        }
        MPI_Finalize();
    }
```

ПРИЛОЖЕНИЕ Б
ТАБЛИЦЫ

Таблица 1 - Зависимость времени работы программы от числа процессов.

Количество процессов	Среднее время выполнения (с)
1	0.000021
2	0.000053
3	0.000075
4	0.000093
5	0.000097
6	0.000105
7	0.000127
8	0.000131
16	0.000943
32	0.001766
64	0.005395

ПРИЛОЖЕНИЕ В
ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

При нескольких запусках программы получены подобные результаты:

sun@aleksundr:~/Документы/PA/lb4$ mpiexec -n 6 mpi_lb4

Start proc rang: 4; value: -3.95647

Start proc rang: 0; value: -1.44563

Start proc rang: 1; value: -3.68716

Start proc rang: 5; value: 1.10227

New prog rang 3; value: 1.10227

New prog rang 2; value: -3.95647

New prog rang 1; value: -3.68716

New prog rang 0; value: -1.44563

sun@aleksundr:~/Документы/PA/lb4$ mpiexec -n 3 mpi_lb4

Start proc rang: 0; value: -1.07085

Start proc rang: 2; value: -1.52482

New prog rang 1; value: -1.52482

New prog rang 0; value: -1.07085

sun@aleksundr:~/Документы/PA/lb4$ mpiexec -n 1 mpi_lb4

Start proc rang: 0; value: 4.4019

New prog rang 0; value: 4.4019