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. 3384 _____ Рудаков А.Л.

Преподаватель _____ Татаринов Ю.С.

Санкт-Петербург
2025

Цель работы.

Изучение коллективных операций в MPI и использование их на практике, путем написания параллельных программ, запускаемых на различном числе одновременно работающих процессов

Задание.

Вариант 9.

В каждом процессе дан набор из K + 5 целых чисел, где K — количество процессов. Используя функцию MPI_Reduce для операции MPI_MAXLOC, найти максимальное значение среди элементов данных наборов с одним и тем же порядковым номером и ранг процесса, содержащего это максимальное значение. Вывести в главном процессе вначале все максимумы, а затем — ранги содержащих их процессов.

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 IntInt, для хранения информации о двух значениях int (так как в функции MPI_MAXLOC на выходе MPI_2INT).

Для каждого процесса случайным образом генерируется вектор values, состоящий из k+5 количества значений (k-количество процессов), в который записываются значения и ранг процесса.

Также определяется вектор final_values для выходных значений функции.

В каждом процессе выполняется MPI_Reduce(values.data(), final_values.data(), count, MPI_2INT, MPI_MAXLOC, 0, MPI_COMM_WORLD).

После чего, в процессе 0 производится полученных значений: вначале максимумов, затем рангов их процессов.

Схема Петри представлена на рис.1.

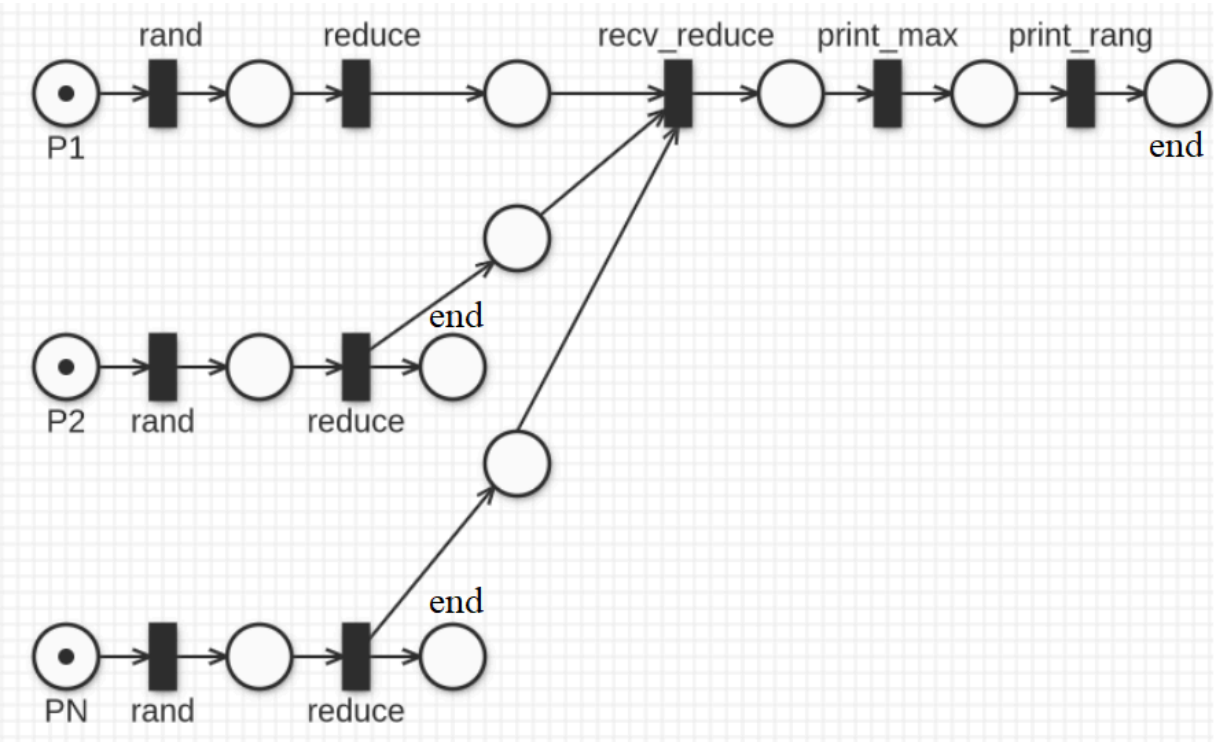

Рисунок 1 — Схема Петри.

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.

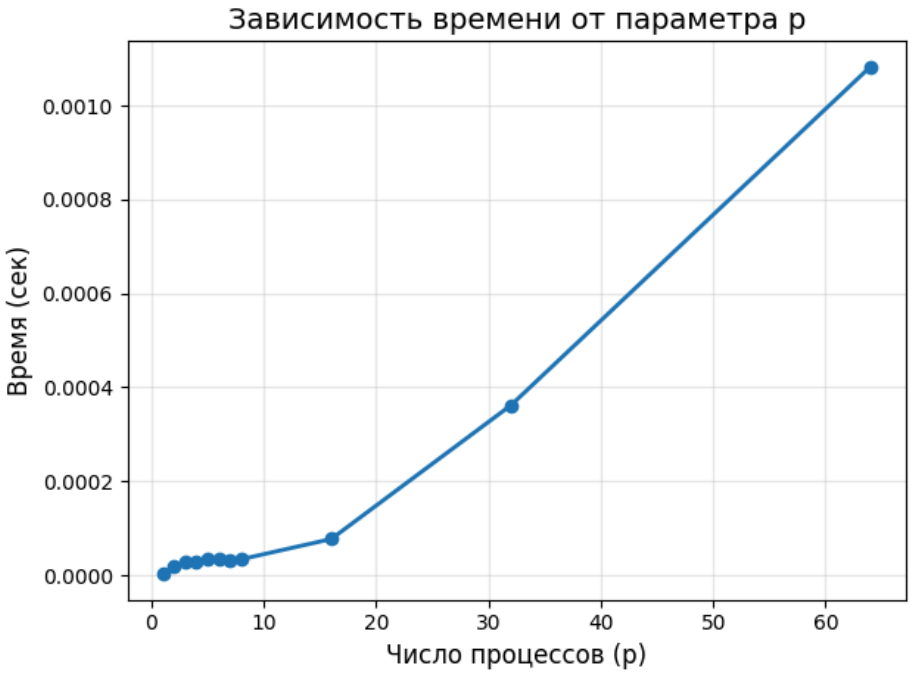

Рисунок 2 - График зависимости времени от числа процессов

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

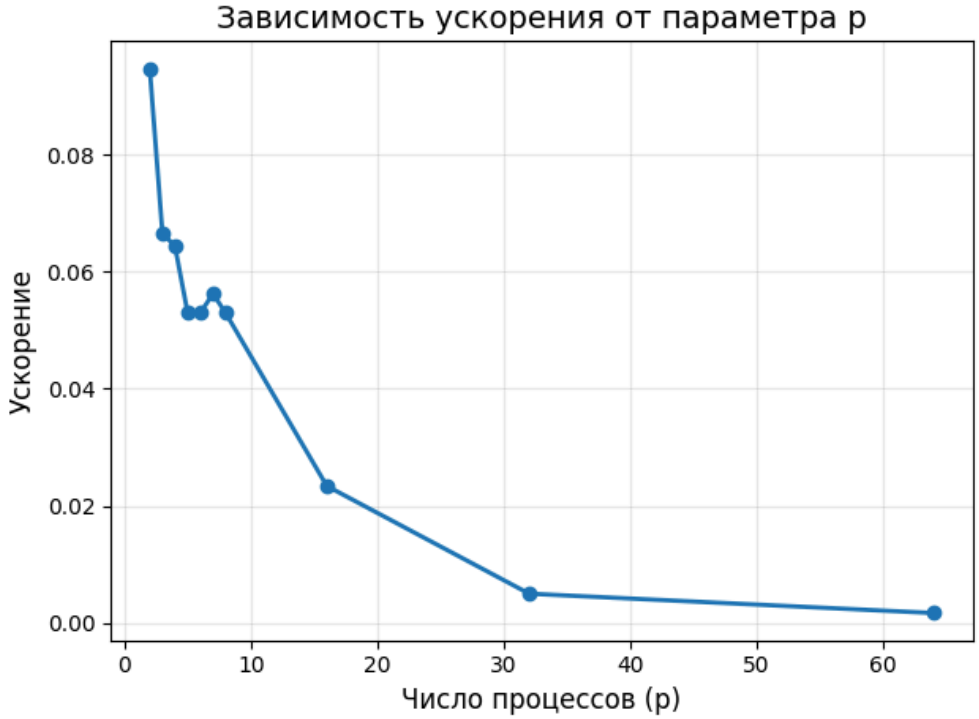

Рисунок 3 - График зависимости ускорения от числа процессов

Из полученных результатов, можно заметить, что при количестве процессов от 2 до 8 время примерно равно, что обуславливается тем, что количество данных внутри каждый раз увеличивается на 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 разбиение на большее количество процессов включается планировщик, который распределяет времена среди потоков, вследствие чего время увеличивается и ускорение уменьшается.

Выводы.

Изучены коллективные операции в MPI, в частности MPI_Reduce, 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_lb3.cpp:

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

struct{
    int value;
    int rang;
} typedef IntInt;

int main(int argc, char** argv) {
    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_int_distribution<> distrib(-100, 100);

    MPI_Init(&argc, &argv);

    int proc_rang, num_proc;
    std::vector<IntInt> values;
    std::vector<IntInt> final_values;

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

    int count = num_proc + 5;
    final_values.resize(count);

    for (int i = 0; i < count; i++){
        IntInt value {distrib(gen), proc_rang};
        values.push_back(value);
    }

    MPI_Reduce(values.data(), final_values.data(), count, MPI_2INT, MPI_MAXLOC, 0, MPI_COMM_WORLD);

    if (proc_rang == 0){
        std::cout << "Max:";
        for (int i = 0; i < count; i++){
```

```
            std::cout << " " << final_values[i].value;
        }
        std::cout << "\nRang:";
        for (int i = 0; i < count; i++){
            std::cout << " " << final_values[i].rang;
        }
        std::cout << '\n';
    }

    MPI_Finalize();
}
```

ПРИЛОЖЕНИЕ Б
ТАБЛИЦЫ

Таблица 1 - Зависимость времени работы программы от числа процессов.

Количество процессов	Среднее время выполнения (с)
1	0.0000018
2	0.000019
3	0.000027
4	0.000028
5	0.000034
6	0.000034
7	0.000032
8	0.000034
16	0.000077
32	0.000362
64	0.001083

ПРИЛОЖЕНИЕ В
ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

При нескольких запусков программы получены подобные результаты:

sun@aleksundr:~/Документы/PA/lb3$ mpiexec -n 6 ./mpi_lb3

Max: 96 81 84 97 44 93 74 56 97 51 99

Rang: 3 5 1 2 4 1 4 3 2 4 2

sun@aleksundr:~/Документы/PA/lb3$ mpiexec -n 3 ./mpi_lb3

Max: 88 81 32 87 90 45 -9 33

Rang: 1 2 1 2 1 0 2 1

sun@aleksundr:~/Документы/PA/lb3$ mpiexec -n 1 ./mpi_lb3

Max: 83 -13 -39 -38 78 100

Rang: 0 0 0 0 0 0