МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №5

по дисциплине «Параллельные алгоритмы»

Тема: Виртуальные топологии

Студент гр. 3384 _____ Рудаков А.Л.

Преподаватель _____ Татаринов Ю.С.

Санкт-Петербург

2025

Цель работы.

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

Задание.

Вариант 4.

Число процессов K является четным: K = 2N, N > 1 В процессах 0 и 1 дано по одному вещественному числу A. Определить для всех процессов декартову топологию в виде матрицы размера N × 2, после чего, используя функцию MPI_Cart_sub, расщепить матрицу процессов на два одномерных столбца (при этом процессы 0 и 1 будут главными процессами в полученных столбцах). Используя одну коллективную операцию пересылки данных, переслать число A из главного процесса каждого столбца во все процессы этого же столбца и вывести полученное число в каждом процессе (включая процессы 0 и 1).

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)*.

Программа требует четное количество процессов для корректной работы. Если условие не выполняется, процесс с рангом 0 выводит сообщение об ошибке.

При помощи функции *MPI_Cart_create(..)* создается 2D сетка процессов с размерами: *num_proc / 2* строк и *2* столбцами. После этого в функции *MPI_Cart_coords(..)* определяются координаты процессов в сетке. При помощи функции *MPI_Cart_sub(..)* создаются вертикальные подкоммуникаторы (колонки), где процессы с одинаковым индексом столбца объединяются в отдельные группы.

После этого в каждой колонке процесс с координатой строки 0 (0 и 1 процесс) отправляют широковещательную рассылку своего value через *MPI_Bcast(..)*. Другие же процессы получают информацию и записывают в *received_value*. После чего в каждом процессе выводиться *received_value*.

Схема Петри представлена на рис.1.

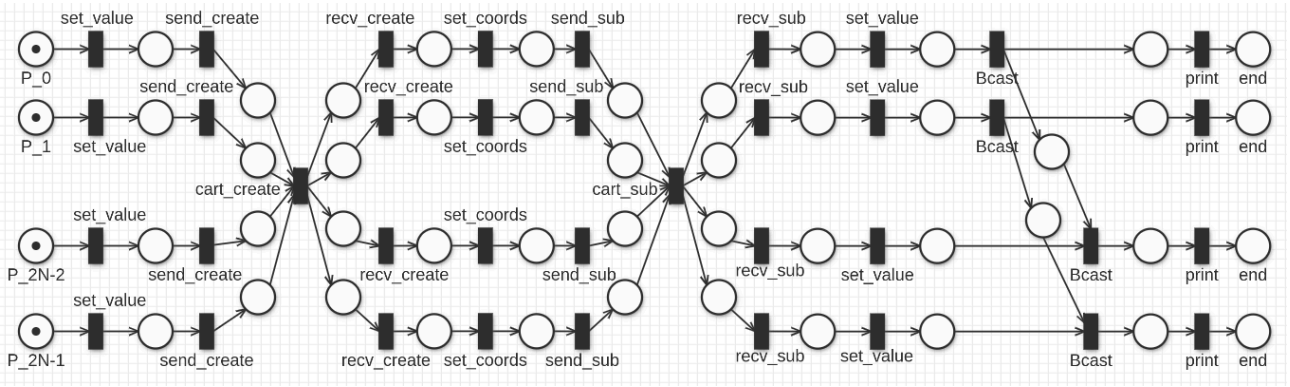

Рисунок 1 — Схема Петри.

Код, написанный 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.

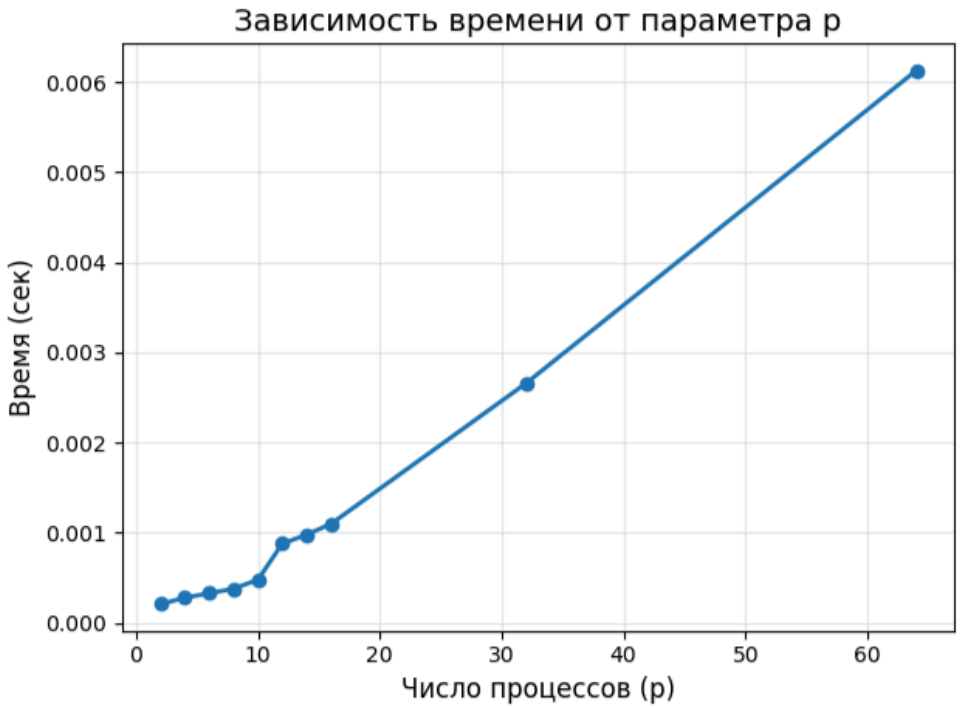

Рисунок 2 - График зависимости времени от числа процессов

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

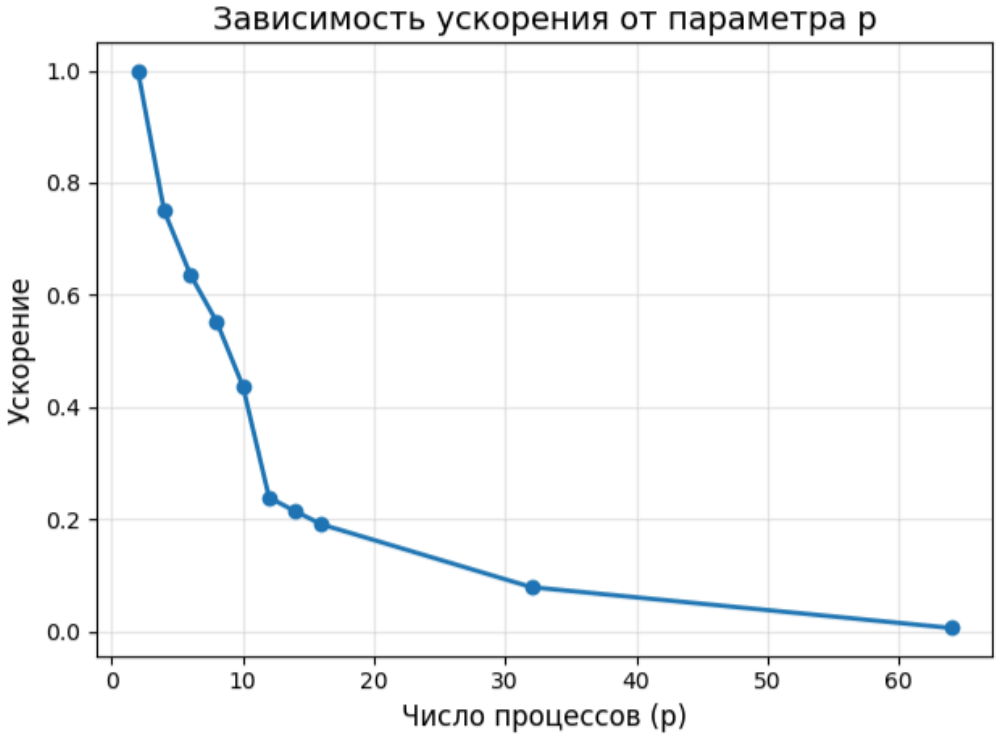

Рисунок 3 - График зависимости ускорения от числа процессов

Из полученных результатов можно заметить, что время растет линейно относительно количества процессов, на которых запускается программа, с небольшим местом перегиба в графике времени (между 10 и 12 процессами) что обуславливается общим количествов потоков на компьютере, на котором

проводилось тестирование. При бОльших количествах процессов время увеличивается быстрее и ускорение падает. Это обуславливается тем, что при разбиении программы на большее количество процессов включается планировщик, который распределяет время среди потоков, вследствие чего время увеличивается и ускорение уменьшается.

Выводы.

Изучены виртуальные топологии в MPI, в частности использована топология решетки, 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_lb5.cpp:

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

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

   MPI_Init(&argc, &argv);

   int proc_rang, num_proc;

   float value, received_value;

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

   if (num_proc % 2 != 0){
      if (proc_rang == 0){
         printf("Error: Number of processes must be even\n");
      }
      MPI_Finalize();
      return 0;
   }

   if (proc_rang == 0){
      value = 10.5f;
   } else if (proc_rang == 1){
      value = 20.7f;
   }

   int dims[2], periods[2], coords[2];
   dims[0] = num_proc / 2;
   dims[1] = 2;
   periods[0] = 0;
   periods[1] = 0;

   int remain_dims[2];
   remain_dims[0] = 1;
   remain_dims[1] = 0;
```

```
    MPI_Comm cart_comm, column_comm;

    MPI_Cart_create(MPI_COMM_WORLD, 2, dims, periods, 0, &cart_comm);

    MPI_Cart_coords(cart_comm, proc_rang, 2, coords);

    MPI_Cart_sub(cart_comm, remain_dims, &column_comm);

    int new_rang, new_size;
    MPI_Comm_rank(column_comm, &new_rang);
    MPI_Comm_size(column_comm, &new_size);

    bool is_root_in_column = (coords[0] == 0);

    if (is_root_in_column){
        MPI_Bcast(&value, 1, MPI_FLOAT, 0, column_comm);
        received_value = value;
    } else {
        MPI_Bcast(&received_value, 1, MPI_FLOAT, 0, column_comm);
    }

    std::cout << "Process number " << proc_rang << " (coords[" << coords[0] << ","
            << coords[1] << "]) received value = " << received_value << "\n";

    MPI_Comm_free(&column_comm);
    MPI_Comm_free(&cart_comm);

    MPI_Finalize();
    return 0;
}
```

ПРИЛОЖЕНИЕ Б
ТАБЛИЦЫ

Таблица 1 - Зависимость времени работы программы от числа процессов.

Количество процессов	Среднее время выполнения (с)
2	0.00021
4	0.00028
6	0.00033
8	0.00038
10	0.00048
12	0.00088
14	0.00098
16	0.00110
32	0.00266
64	0.00613

ПРИЛОЖЕНИЕ В
ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

При нескольких запусках программы получены подобные результаты:

sun@aleksundr:~/Документы/PA/lb5$ mpiexec --oversubscribe -n 4 mpi_lb5

Process number 0 (coords[0,0]) received value = 10.5

Process number 1 (coords[0,1]) received value = 20.7

Process number 2 (coords[1,0]) received value = 10.5

Process number 3 (coords[1,1]) received value = 20.7

sun@aleksundr:~/Документы/PA/lb5$ mpiexec --oversubscribe -n 16 mpi_lb5

Process number 10 (coords[5,0]) received value = 10.5

Process number 12 (coords[6,0]) received value = 10.5

Process number 14 (coords[7,0]) received value = 10.5

Process number 15 (coords[7,1]) received value = 20.7

Process number 0 (coords[0,0]) received value = 10.5

Process number 1 (coords[0,1]) received value = 20.7

Process number 2 (coords[1,0]) received value = 10.5

Process number 3 (coords[1,1]) received value = 20.7

Process number 5 (coords[2,1]) received value = 20.7

Process number 8 (coords[4,0]) received value = 10.5

Process number 4 (coords[2,0]) received value = 10.5

Process number 6 (coords[3,0]) received value = 10.5

Process number 7 (coords[3,1]) received value = 20.7

Process number 11 (coords[5,1]) received value = 20.7

Process number 13 (coords[6,1]) received value = 20.7

Process number 9 (coords[4,1]) received value = 20.7

sun@aleksundr:~/Документы/PA/lb5$ mpiexec --oversubscribe -n 2 mpi_lb5

Process number 0 (coords[0,0]) received value = 10.5

Process number 1 (coords[0,1]) received value = 20.7