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

ОТЧЕТ

по лабораторной работе №2
по дисциплине «Параллельные алгоритмы»
Тема: Использование функций обмена данными «точка-точка» в библиотеки MPI.

Студент гр. 3384 _____ Рудаков А.Л.

Преподаватель _____ Татаринов Ю.С.

Санкт-Петербург
2025

Цель работы.

Изучение функций обмена данными в MPI и использование их на практике, путем написания параллельной программы, запускаемой на различном числе одновременно работающих процессов

Задание.

Вариант 9:.

Игра с ведущим (съедобное − несъедобное).

Процесс 0 поочередно посылает остальным процессам сообщение. Получив сообщение, процесс-приемник информирует об этом процесс 0

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)*.

Далее выполнено разделение работы процессов, в зависимости от их рангов. Если ранг процесса равен 0, то он поочередно для каждого процесса случайным образом выбирает из массива значений слово, которое после отправляет текущему процессу, при помощи функции MPI_Send(word.c_str(), word.length() + 1, MPI_CHAR, i, 0, MPI_COMM_WORLD), после чего ожидает от него ответного сообщения, в котором будет указано, съедобное слово или нет, при помощи MPI_Recv(response, 50, MPI_CHAR, i, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE).

Если же ранг процесса не равен 0, то он принимает сообщение от процесса с рангом 0, через функцию MPI_Recv(response, 50, MPI_CHAR, 0, 0,

MPI_COMM_WORLD, MPI_STATUS_IGNORE), после чего определяет, съедобное слово или нет и отправляет свой вердикт обратно процессору номер 0, через функцию MPI_Send(word.c_str(), word.length() + 1, MPI_CHAR, 0, 0, MPI_COMM_WORLD).

Сеть Петри представлена на рис.1.

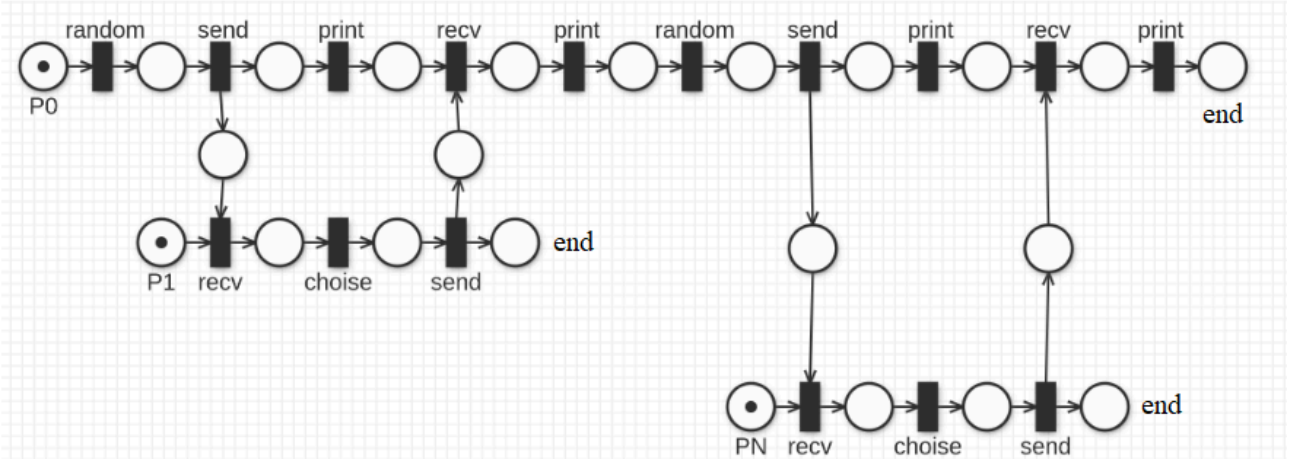

Рисунок 1 — Сеть Петри.

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

Примеры работы программы см. Приложени В.

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

Табл.1. см. Приложение Б.

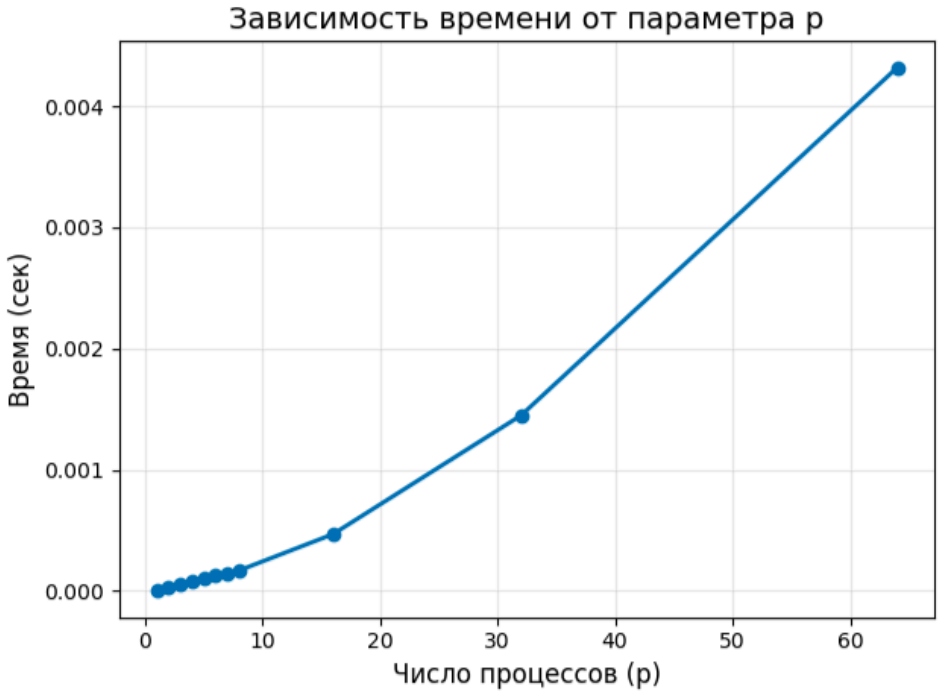

Рисунок 2 - График зависимости времени от числа процессов

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

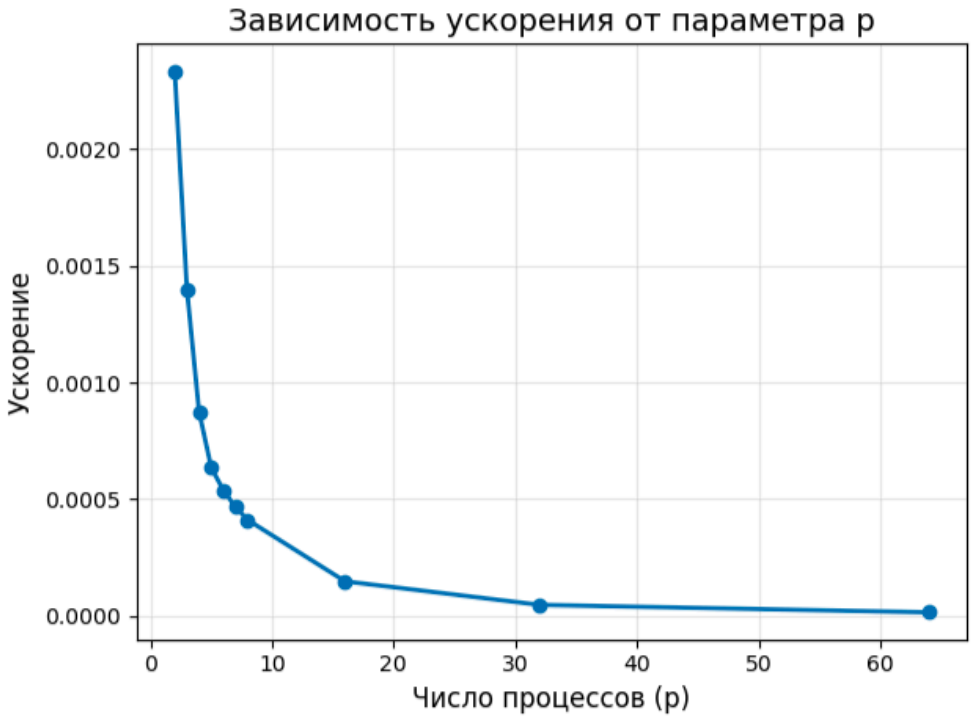

Рисунок 3 - График зависимости ускорения от числа процессов

Исходя из полученных результатов можно заметить, что время растет линейно с увеличением количества процессов. Это связано с тем, что процесс с рангом 0 отправляет одному процессу сообщение, после чего ожидает от него же ответ, только после этого начинает проделывать то же самое с последующими процессами. В связи с этим при увеличении числа процессов на n, на n увеличивается и количество таких итераций, поэтому зависимость линейная.

Ускорение также уменьшается при увеличении числа процессов, что обусловлено теми же самыми причинами.

Выводы.

Изучены функции обмена в MPI, на основании которых написана параллельная программа, запускаемая на различном числе одновременно работающих процессов, реализующая игру “съедобное-несъедобное” между процессами. Проверена работоспособность программы на различном числе процессов, а также замерено время ее работы и ускорение. Выяснено, что

данная программа не “распараллеливается” в силу того, что процесс с рангом 0 поочередно отправляет сообщения другим процессам и ожидает от них ответов.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД

Файл mpi_lb2.cpp:

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

int main(int argc, char** argv){
  std::vector<std::vector<std::string>> values = {{"яблоко", "каша", "помидорка"}, {"гвоздь","телефон","бетон"}};
    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_int_distribution<> distrib_category(0,1);
    std::uniform_int_distribution<> distrib_value(0,2);

    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);

     if (num_proc > 1){
       if (proc_rang == 0){
          for (int i = 1; i < num_proc; i++){
             int category = distrib_category(gen);
             int value = distrib_value(gen);
             std::string word = values[category][value];

             MPI_Send(word.c_str(), word.length() + 1, MPI_CHAR, i, 0, MPI_COMM_WORLD);
             std::cout << "0 proccess send \"" << word << "\" to proccess " << i << "\n";

             char response[50];
             MPI_Recv(response, 50, MPI_CHAR, i, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
             std::cout << i << " proccess send \"" << response << "\" to 0 proccess\n";
          }
       }
       else{
```

```
            char response[50];
            MPI_Recv(response, 50, MPI_CHAR, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);

            auto it = std::find(values[0].begin(), values[0].end(), response);

            std::string word = (it !=values[0].end()) ? "съедобное" : "несъедобное";
            MPI_Send(word.c_str(), word.length() + 1, MPI_CHAR, 0, 0, MPI_COMM_WORLD);
        }
    }

    MPI_Finalize();
}
```

ПРИЛОЖЕНИЕ Б
ТАБЛИЦЫ

Таблица 1 - Зависимость времени работы программы от числа процессов.

Количество процессов	Среднее время выполнения (с)
1	0.00000007
2	0.00003
3	0.00005
4	0.00008
5	0.00011
6	0.00013
7	0.00015
8	0.00017
16	0.00047
32	0.00145
64	0.00432

ПРИЛОЖЕНИЕ В
ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

Примеры работы программы:

0 proccess send "бетон" to proccess 1

1 proccess send "несъедобное" to 0 proccess

0 proccess send "каша" to proccess 2

2 proccess send "съедобное" to 0 proccess

0 proccess send "помидорка" to proccess 3

3 proccess send "съедобное" to 0 proccess

0 proccess send "помидорка" to proccess 4

4 proccess send "съедобное" to 0 proccess

0 proccess send "бетон" to proccess 5

5 proccess send "несъедобное" to 0 proccess

0 proccess send "гвоздь" to proccess 6

6 proccess send "несъедобное" to 0 proccess

0 proccess send "телефон" to proccess 7

7 proccess send "несъедобное" to 0 proccess

...

0 proccess send "бетон" to proccess 1

1 proccess send "несъедобное" to 0 proccess

0 proccess send "каша" to proccess 2

2 proccess send "съедобное" to 0 proccess

0 proccess send "гвоздь" to proccess 3

3 proccess send "несъедобное" to 0 proccess