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

ОТЧЕТ

по лабораторной работе №0

по дисциплине по дисциплине «Параллельные алгоритмы»

Тема: Запуск параллельной программы на различном числе одновременно работающих процессов, упорядочение вывода результатов

Студент гр. 1384 Усачева Д. В.

Преподаватель Татаринов Ю. С.

Санкт-Петербург

2023

Цель

Ознакомиться с библиотекой MPI, написав параллельную программу с последующим анализом.

Задание

Запустить программу на 1,2 … N процессах несколько раз. Проанализировать порядок вывода сообщений на экран. Вывести правило, определяющее порядок вывода сообщений. Модифицировать программу таким образом, чтобы порядок вывода сообщений на экран соответствовал номеру соответствующего процесса.

1. Написать параллельную программу MPI, где каждый процесс определяет свой ранг MPI_Comm_rank (MPI_COMM_WORLD, &ProcRank);, после чего действия в программе разделяются. Все процессы, кроме процесса с рангом 0 else, передают значение своего ранга нулевому процессу MPI_Send (&ProcRank, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);. Процесс с рангом 0 if (ProcRank == 0){...} сначала печатает значение своего ранга printf ("\n Hello from process %3d", ProcRank);, а далее последовательно принимает сообщения с рангами процессов MPI_Recv(&RecvRank, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &Status); и также печатает их значения printf("\n Hello from process %3d", RecvRank); . При этом важно отметить, что порядок приема сообщений заранее не определен и зависит от условий выполнения параллельной программы (более того, этот порядок может изменяться от запуска к запуску).

2. Запустить программу на 1,2 … N процессах несколько раз.

3. Проанализировать порядок вывода сообщений на экран. Вывести правило, определяющее порядок вывода сообщений.

4. Модифицировать программу таким образом, чтобы порядок вывода сообщений на экран соответствовал номеру соответствующего процесса.

Выполнение работы

Для выполнения поставленной задачи написана программа на языке C, код которой представлен ниже в листинге 1.

Листинг 1 — Код программы lab0.c

```
#include "mpi.h"
#include "stdio.h"
#include "stdlib.h"

int main ( int argc, char *argv[] ) {
    int ProcNum, ProcRank, RecvRank;
    double t1, t2, dt;
    MPI_Init ( &argc, &argv );
    t1 = MPI_Wtime();
    MPI_Comm_size ( MPI_COMM_WORLD, &ProcNum);
    MPI_Comm_rank ( MPI_COMM_WORLD, &ProcRank);
    if (ProcRank == 0){
        MPI_Status Status;
           printf ("Hello from process %3d\n", ProcRank);
           for (int i = 1; i <= ProcNum - 1; i++){
            MPI_Recv(&RecvRank,   1,   MPI_INT,   MPI_ANY_SOURCE,
MPI_ANY_TAG, MPI_COMM_WORLD, &Status);
                printf("Hello from process %3d\n", RecvRank);
           }
     }
    else{
        MPI_Send(&ProcRank, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
    }
    t2 = MPI_Wtime();
    MPI_Finalize();
    dt = t2 - t1;
    if(ProcRank==0){
        printf("Process  number  %3d  finished  in  %3f\n",ProcRank,
dt);
    }

return 0;
}
```

Ниже в листинге 2 представлен вывод программы.

Листинг 2 — Выполнение программы lab0.c для разного количества процессов

```
Hello from process   0
Process number   0 finished in 0.000056

Hello from process   0
Hello from process   1
Process number   0 finished in 0.000139

Hello from process   0
Hello from process   1
Hello from process   2
Process number   0 finished in 0.000093

Hello from process   0
Hello from process   1
Hello from process   2
Hello from process   3
Process number   0 finished in 0.000189
```

```
Hello from process   0
Hello from process   1
Hello from process   2
Hello from process   3
Hello from process   4
Hello from process   6
Hello from process   5
Hello from process   7
Process number   0 finished in 0.000107

Hello from process   0
Hello from process   1
Hello from process   3
Hello from process   4
Hello from process   5
Hello from process   6
Hello from process   7
Hello from process   9
Hello from process  12
Hello from process  13
Hello from process   2
Hello from process   8
Hello from process  10
Hello from process  11
Hello from process  14
Hello from process  15
Process number   0 finished in 0.000346
```

Данный алгоритм может быть представлен в виде следующей сети Петри (см. рисунок 1).

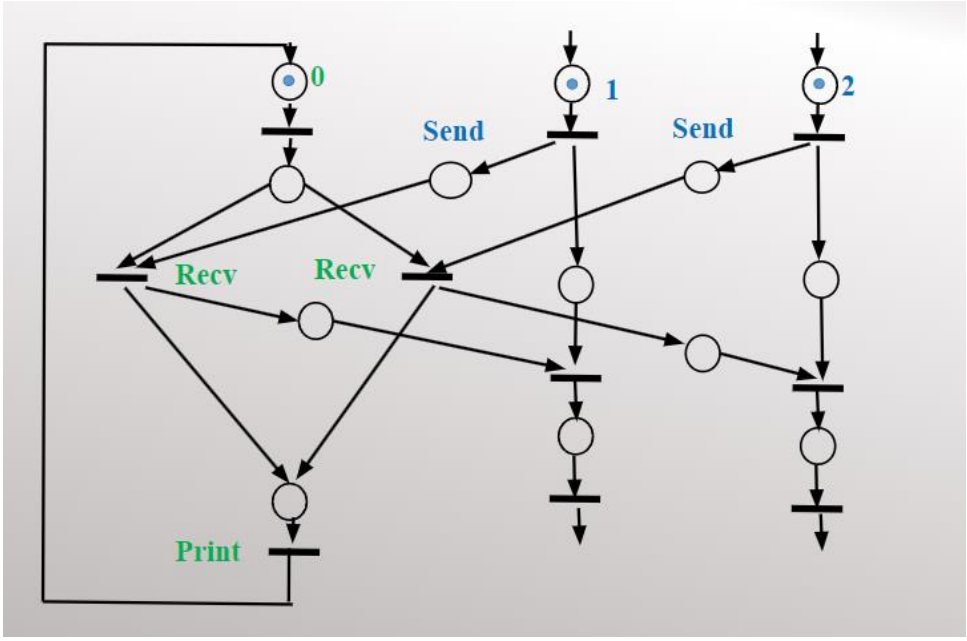

Рисунок 1 – Сеть Петри

Из полученных результатов можно сделать вывод, что сначала всегда идет запись нулевого процесса, затем несколько процессов идут по порядку,

далее в порядке близком к случайному. То есть, если запустить несколько процессов, то вывод будет зависеть от порядка получения сообщений нулевым процессом. Ниже приведена таблица среднего времени выполнения для количества процессов от 1 до 64 (см. таблица 1).

Таблица 1 — Среднее время выполнения процессов.

Количество процессов	Среднее время на выполнение(мс)
1	0.045
2	0.059
4	0.066
8	0.156
16	0.342
32	0.764
64	1.312

Далее приведен график зависимости времени выполнения от количества процессов (см. рисунок 2).

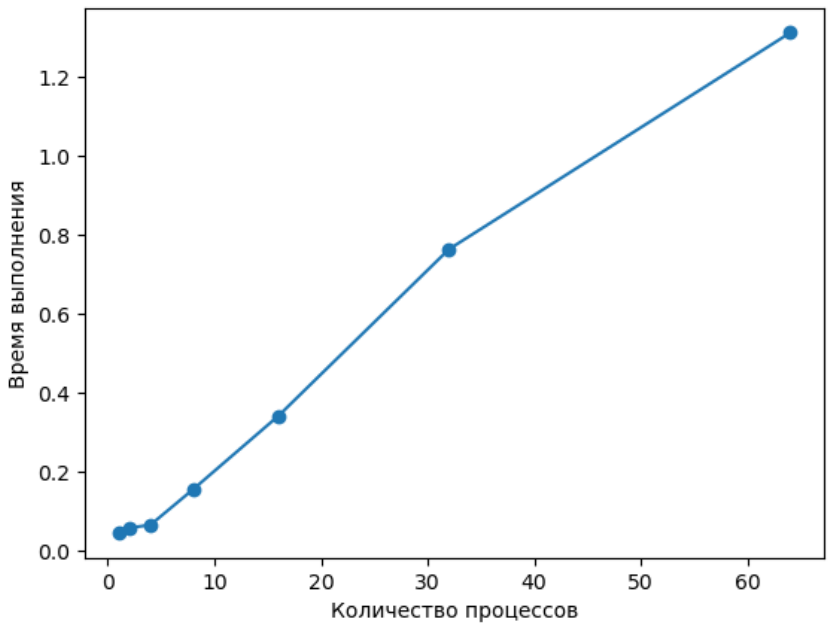

Рисунок 2 — График зависимости времени выполнения от числа процессов

Ускорение времени работы программы можно вычислить по формуле:

$$S_p(n) = T_1(n)/T_p(n)$$

Результаты представлены на графике ниже (см. рисунок 3).

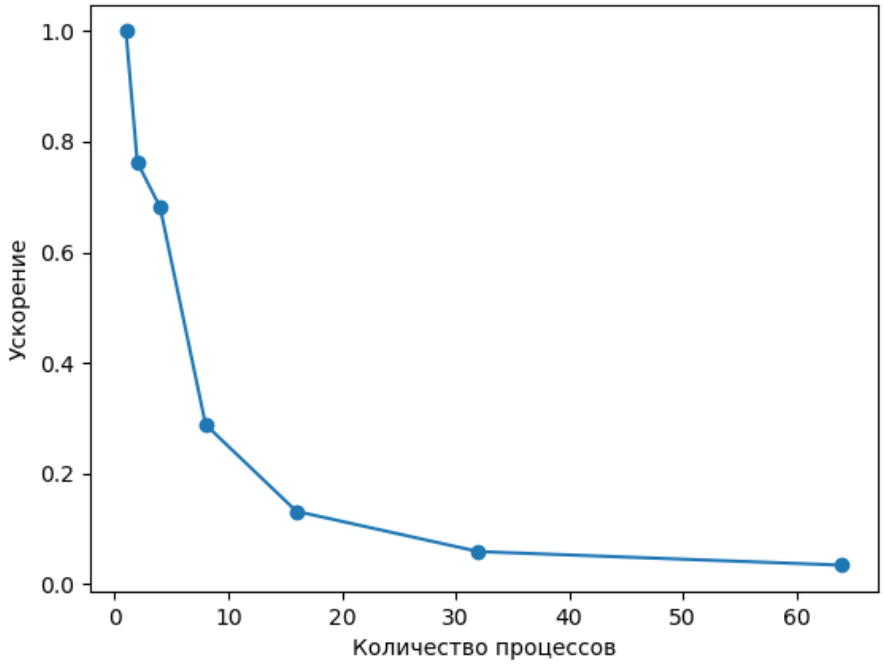

Рисунок 3 — График зависимости ускорения от числа процессов

Как видно из графика ускорения, при увеличении количества процессов программа замедляется.

Затем программу нужно было модифицировать таким образом, чтобы сообщения от процессов выводились согласно их рангу. Код программы представлен в листинге 3.

Листинг 3 — Код модифицированной программы

```
#include <mpi.h>
#include "stdio.h"
#include "stdlib.h"

int main(int argc, char* argv[]) {
      int procNum, ProcRank,RecvRank;
      double start, end;
      MPI_Init(&argc, &argv);
      start = MPI_Wtime();
      MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);
      MPI_Comm_size(MPI_COMM_WORLD, &procNum);
      if (ProcRank != 0)
      {
            MPI_Send(&ProcRank, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
      }
      else
      {
            MPI_Status Status;
            printf("Greetings from process rank % 3d\n", ProcRank);
            for (int i = 1; i <= procNum - 1; i++)
```

```
                {
                        MPI_Recv(&RecvRank, 1, MPI_INT, i, MPI_ANY_TAG,
MPI_COMM_WORLD, &Status);
                        printf("Greetings from rank % 3d \n", RecvRank);
                }
        }
        end = MPI_Wtime();
        MPI_Finalize();
        printf("Process number %3d finished in %3f\n",ProcRank,end -
start);
        return 0;
}
```

Ниже представлен вывод программы.

Листинг 4 — Вывод модифицированной программы.

```
Greetings from process rank   0
Greetings from rank   1
Greetings from rank   2
Greetings from rank   3
Greetings from rank   4
Greetings from rank   5
Greetings from rank   6
Greetings from rank   7
Greetings from rank   8
Greetings from rank   9
Greetings from rank  10
Greetings from rank  11
Greetings from rank  12
Greetings from rank  13
Greetings from rank  14
Greetings from rank  15
```

Далее приведена таблица среднего времени (см. таблица 2).

Таблица 2 — Среднее время выполнения количества процессов.

Количество процессов	Среднее время на выполнение(мс)
1	0.045
2	0.051
4	0.066
8	0.189
16	0.452
32	1.15
64	3.252

Ниже указан график зависимости времени выполнения от количества процессов (см. рисунок 4).

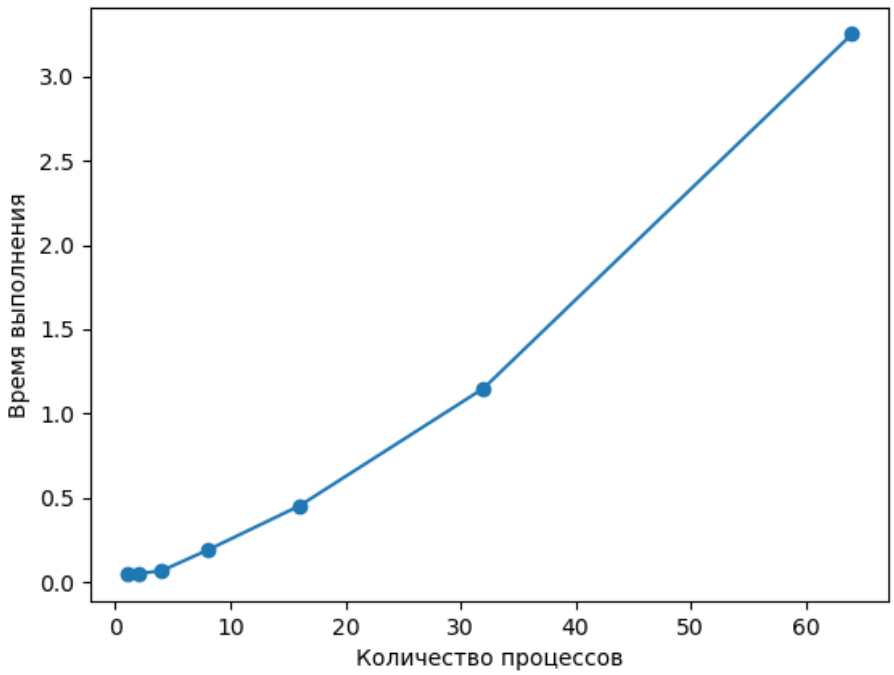

Рисунок 4 — График зависимости времени выполнения от числа процессов

Выводы

В ходе выполнения лабораторной работы были получены навыки компиляции, разработки и запуска параллельной программы, использующей библиотеку MPI. Была выполнена модификация программы таким образом, чтобы порядок вывода сообщений на экран соответствовал номеру соответствующего процесса. Также построены графики времени работы программы и ускорения от увеличения количества процессов, которые указывают на падение эффективности программы для большего числа процессов. Время работы программы для упорядоченного получения и случайного получения сообщений отличается незначительно. Вероятно, небольшую задержку для модифицированной программы можно обосновать тем, что процессам, которые раньше отправят сообщение, придется ждать, пока нулевой процесс обработает сообщения от других процессов с меньшим рангом.