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

ОТЧЕТ

по лабораторной работе №1

по дисциплине «Параллельные алгоритмы»

Тема: Запуск параллельной программы на различном числе одновременно работающих процессов

Студентка гр. 2384 _____ Соц Е.А.

Преподаватель _____ Татаринов Ю.С

Санкт-Петербург
2024

Задание

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. Построить график времени работы программы в зависимости от числа запущенных процессов от 1 до 16 Размер шага – например, 4
5. Построить график ускорения/замедления работы программы.
6. Модифицировать программу таким образом, чтобы порядок вывода сообщений на экран соответствовал номеру соответствующего процесса.
7. Нарисовать сеть Петри для двух вариантов MPI программы.

Выполнение работы

1. Код разработанной программы:

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

int main(int argc, char** argv) {
    int ProcRank;
    int RecvRank;
    int ProcSize;

    // Инициализация MPI
    MPI_Init(&argc, &argv);

    // Получение ранга текущего процесса
    MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);
    MPI_Comm_size(MPI_COMM_WORLD, &ProcSize);

       // Если это не процесс с рангом 0, отправляем свое
значение ранга процессу с рангом 0
    if (ProcRank != 0) {
                    MPI_Send(&ProcRank,  1,  MPI_INT,  0,  0,
MPI_COMM_WORLD);
    }

    // Процесс с рангом 0 выводит свое значение ранга
    if (ProcRank == 0) {
        printf("Hello from process %3d\n", ProcRank);

        // Принимаем сообщения от других процессов
        for(int i = 1; i < ProcSize; i++){
           MPI_Recv(&RecvRank,  1,  MPI_INT,  MPI_ANY_SOURCE,
MPI_ANY_TAG,
                            MPI_COMM_WORLD, MPI_STATUS_IGNORE);
           printf("Hello from process %3d\n", RecvRank);
        }
    }

    // Завершение MPI программы
    MPI_Finalize();
    return 0;
}
```

- Инициализация MPI и получение информации о процессе:

Определяется ранг процесса (ProcRank) и общее количество процессов (ProcSize).

- Отправка информации о ранге процесса:

Если процесс не имеет ранга 0, он отправляет свое значение ранга на ранг 0.

- Обработка сообщений от других процессов:

Процесс с рангом 0 принимает сообщения от всех остальных процессов и выводит их значения ранга.

Данный алгоритм демонстрирует базовые концепции коммуникации в MPI, включая отправку и прием сообщений между процессами в распределенной системе.

2. Листинг запущенной программы на каком-то количестве процессов несколько раз:

```
katya@katya:~/ПА$ mpirun -np 2 ./lab1
Hello from process   0
Hello from process   1
katya@katya:~/ПА$ mpirun -np 2 ./lab1
Hello from process   0
Hello from process   1
katya@katya:~/ПА$ mpirun -np 2 ./lab1
Hello from process   0
Hello from process   1
katya@katya:~/ПА$ mpirun -np 6 ./lab1
Hello from process   0
Hello from process   5
Hello from process   2
Hello from process   3
Hello from process   1
Hello from process   4
katya@katya:~/ПА$ mpirun -np 6 ./lab1
Hello from process   0
Hello from process   3
Hello from process   2
Hello from process   5
Hello from process   1
Hello from process   4
katya@katya:~/ПА$ mpirun -np 6 ./lab1
Hello from process   0
Hello from process   3
Hello from process   2
Hello from process   5
Hello from process   1
Hello from process   4
katya@katya:~/ПА$ mpirun -np 8 ./lab1
Hello from process   0
Hello from process   4
Hello from process   6
Hello from process   5
```

```
Hello from process   1
Hello from process   2
Hello from process   7
Hello from process   3
katya@katya:~/ПА$ mpirun -np 8 ./lab1
Hello from process   0
Hello from process   4
Hello from process   6
Hello from process   2
Hello from process   7
Hello from process   3
Hello from process   5
Hello from process   1
katya@katya:~/ПА$ mpirun -np 8 ./lab1
Hello from process   0
Hello from process   4
Hello from process   2
Hello from process   6
Hello from process   5
Hello from process   1
Hello from process   3
Hello from process   7
katya@katya:~/ПА$ mpirun -np 16 ./lab1
Hello from process   0
Hello from process  15
Hello from process  11
Hello from process  13
Hello from process   5
Hello from process   9
Hello from process   3
Hello from process  14
Hello from process  12
Hello from process   6
Hello from process  10
Hello from process   7
Hello from process   1
Hello from process   8
Hello from process   2
Hello from process   4
katya@katya:~/ПА$ mpirun -np 16 ./lab1
Hello from process   0
Hello from process  11
Hello from process   3
Hello from process  14
Hello from process   6
Hello from process  15
Hello from process   7
Hello from process  13
Hello from process   5
Hello from process   9
Hello from process  12
Hello from process   4
Hello from process   8
Hello from process   2
Hello from process   1
Hello from process  10
```

```
katya@katya:~/ПА$ mpirun -np 16 ./lab1
Hello from process   0
Hello from process   8
Hello from process  12
Hello from process   4
Hello from process  10
Hello from process   2
Hello from process   9
Hello from process   1
Hello from process  11
Hello from process   3
Hello from process  15
Hello from process   7
Hello from process  14
Hello from process   6
Hello from process  13
Hello from process   5
```

3. Анализируя порядок вывода сообщений на экран при различных запусках программы с различным количеством процессов, можно увидеть, что первым всегда выводится сообщение 0 процесса, а порядок последующих сообщений не определен однозначно и меняется при разных запусках.

Процесс с рангом 0 использует цикл для принятия сообщений от всех остальных процессов с помощью MPI_Recv(). MPI_Recv() работает в режиме ожидания сообщения от любого источника (MPI_ANY_SOURCE), поэтому порядок приема сообщений не определен.

Это поведение демонстрирует распределенную природу параллельного вычисления и отсутствие строгой последовательности операций в многопроцессорных системах.

4. Для изменения размера входных данных был написан скрипт time.sh, который запускает программу.

   Код скрипта time.sh:

```
#!/bin/bash

mpicc -o time time_and_size.c -lmpi -lm

for num_processes in 1 4 8 16; do
    echo "Running with $num_processes processes"

    for data_size in 1 10 100 1000; do
```

```
        echo "  Data size: $data_size"

        mpirun -np $num_processes ./time $data_size

        # Оставляем небольшую паузу между запусками
        sleep 1
    done

    echo ""
done
```

Измененная программа time_and_size.c

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


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


    MPI_Init(&argc, &argv);
    int procRank, procSize;

    MPI_Comm_rank(MPI_COMM_WORLD, &procRank);
    MPI_Comm_size(MPI_COMM_WORLD, &procSize);

    int dataSize = 1;
    if (argc > 1)
    {
        dataSize = atoi(argv[1]);
    }

    int *data = (int *)malloc(dataSize * sizeof(int));

    double start, end, time;

    if (procRank == 0)
    {
        start = MPI_Wtime();

        for (int i = 1; i < procSize; i++)
        {
            MPI_Recv(data, dataSize, MPI_INT, MPI_ANY_SOURCE,
MPI_ANY_TAG, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        }

        end = MPI_Wtime();
        time = end - start;
        printf("Data: %d, Time: %.8f\n", dataSize, time);
    }
    else
    {
        for (int i = 0; i < dataSize; i++)
```

```
        {
            data[i] = procRank;
        }

                MPI_Send(data, dataSize, MPI_INT, 0, 0,
MPI_COMM_WORLD);
    }

    free(data);
    MPI_Finalize();
    return 0;
}
```

Листинг программы:

```
katya@katya:~/ПА$ ./time.sh
Running with 1 processes
Data: 1, Time: 0.00000021
Data: 10, Time: 0.00000026
Data: 100, Time: 0.00000024
Data: 1000, Time: 0.00000020

Running with 4 processes
Data: 1, Time: 0.00004335
Data: 10, Time: 0.00004208
Data: 100, Time: 0.00001925
Data: 1000, Time: 0.00004013

Running with 8 processes
Data: 1, Time: 0.00009447
Data: 10, Time: 0.00008319
Data: 100, Time: 0.00010022
Data: 1000, Time: 0.00008324

Running with 16 processes
Data: 1, Time: 0.00251192
Data: 10, Time: 0.00107872
Data: 100, Time: 0.00102182
Data: 1000, Time: 0.00408178
```

Таким образом получаем графики, показанные на рис.1 и рис.2:

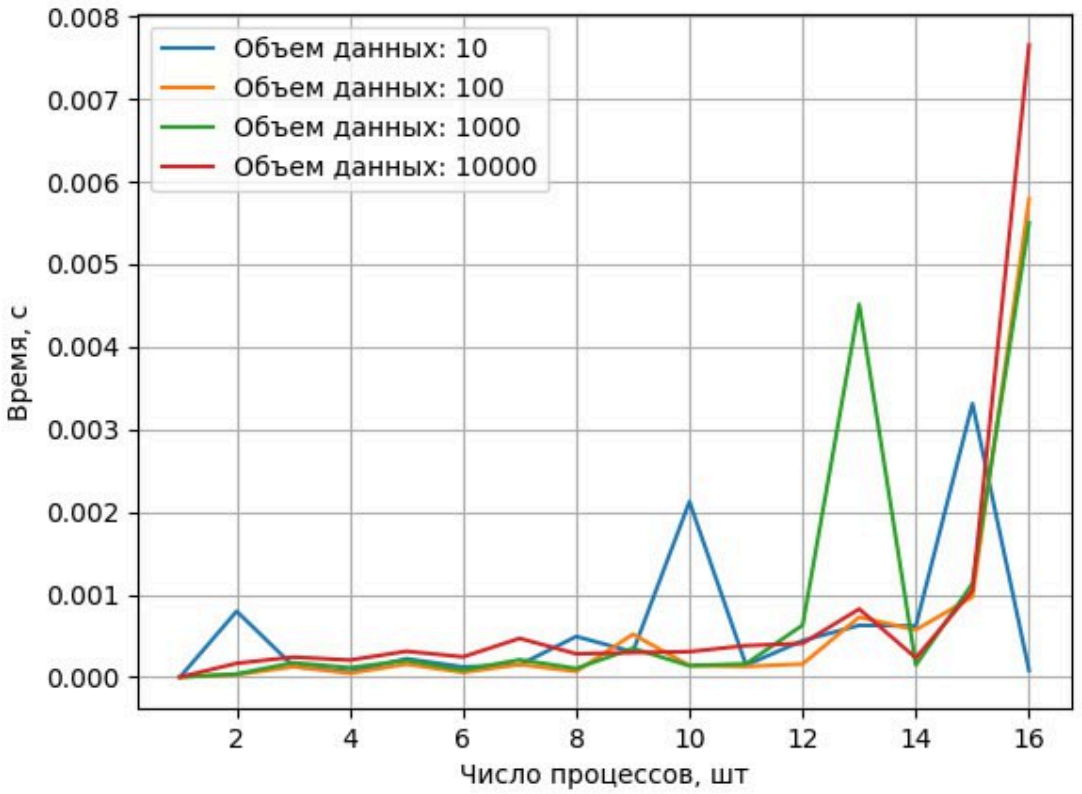

Рисунок 1 – Изменение времени от количества процессов для разных входных данных

Из рисунка видно, что время работы увеличивается при увеличении количества процессов. Это можно объяснить очередью у нулевого процесса сообщений других процессов. Таким образом, чем больше количество процессов, тем больше время ожидания от них сообщения.

Также из рисунка видно, что происходит рост времени от размера объема данных. Это можно объяснить тем, что при отправке данных большего размера нужно больше ресурсов.

5. Анализ ускорения/замедления работы программы

Для того, чтобы провести анализ ускорения или замедления работы программы, построим график, используя формулу ускорения: $Sp(n) = T1(n)/Tp(n)$, где $T1(n)$ – среднее время выполнения алгоритма на одном процессе, $Tp(n)$ - время выполнения алгоритма на 1,4,8 и 16 процессах.

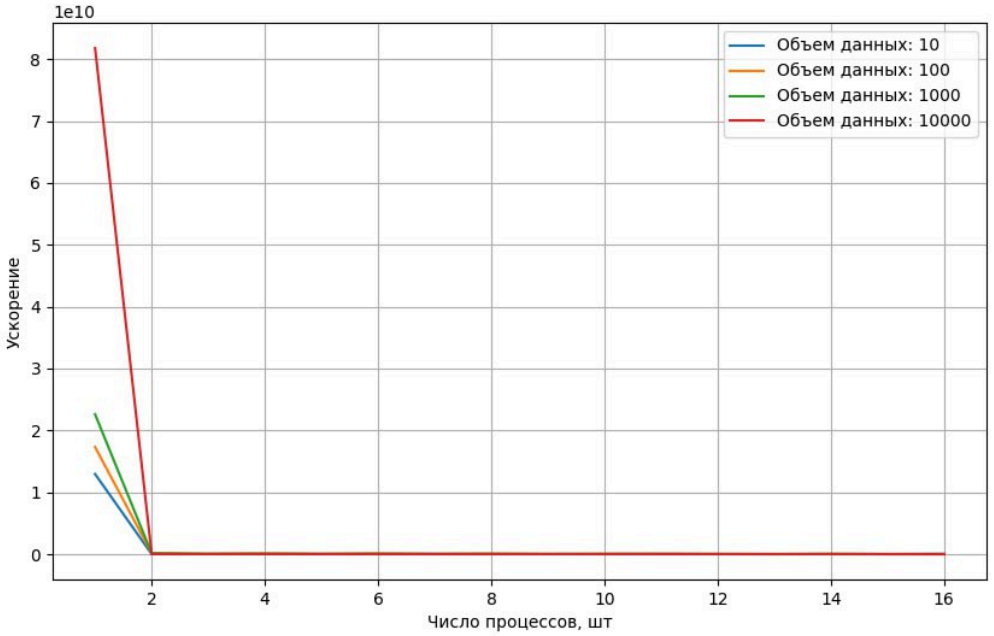

Рисунок 2 – Ускорение/замедление работы программы

Из рисунка видно, что время выполнения параллельной программы бесконечно велико по сравнению с последовательной программой (функция графика стремится к нулю). Можно заметить, что такое происходит даже на данных малого размера.

6. Программа, где порядок вывода сообщений на экран соответствует номеру соответствующего процесса от изначальной программы отличается незначительно: теперь нужно, чтобы в MPI_Recv поступало сообщение не от любого процесса, а чтобы они шли “по порядку”. Таким образом в int MPI_Recv(void *buf, int count, MPI_Datatype type, int source, int tag, MPI_Comm comm, MPI_Status *status) source – порядковый номер процесса.

Программа end.c:

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

int main(int argc, char** argv) {
    int ProcRank;
    int RecvRank;
    int ProcSize;
```

```
    // Инициализация MPI
    MPI_Init(&argc, &argv);

    // Получение ранга текущего процесса
    MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);
    MPI_Comm_size(MPI_COMM_WORLD, &ProcSize);

     // Если это не процесс с рангом 0, отправляем свое значение
ранга процессу с рангом 0
    if (ProcRank != 0) {
        MPI_Send(&ProcRank, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
    }

    // Процесс с рангом 0 выводит свое значение ранга
    if (ProcRank == 0) {
        printf("Hello from process %3d\n", ProcRank);

        // Принимаем сообщения от других процессов
        for(int i = 1; i < ProcSize; i++){
           MPI_Recv(&RecvRank, 1, MPI_INT, i, MPI_ANY_TAG,
                        MPI_COMM_WORLD, MPI_STATUS_IGNORE);
           printf("Hello from process %3d\n", RecvRank);
        }
    }

    // Завершение MPI программы
    MPI_Finalize();
    return 0;
}
```

7. Полученные сети Петри

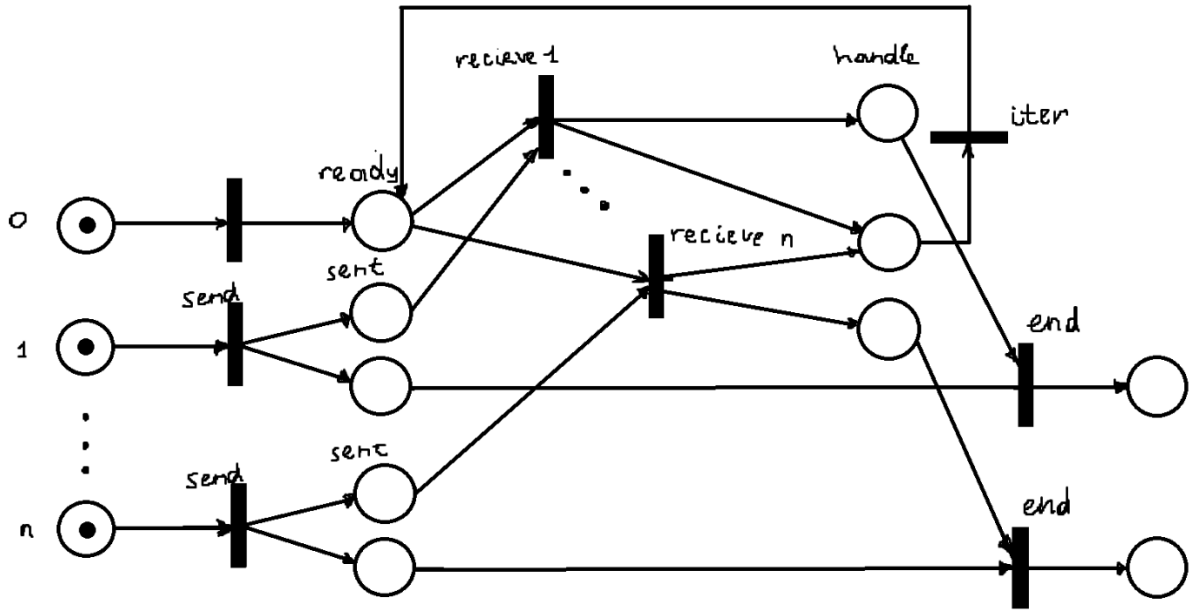

Рисунок 3 – Сеть Петри для первой программы

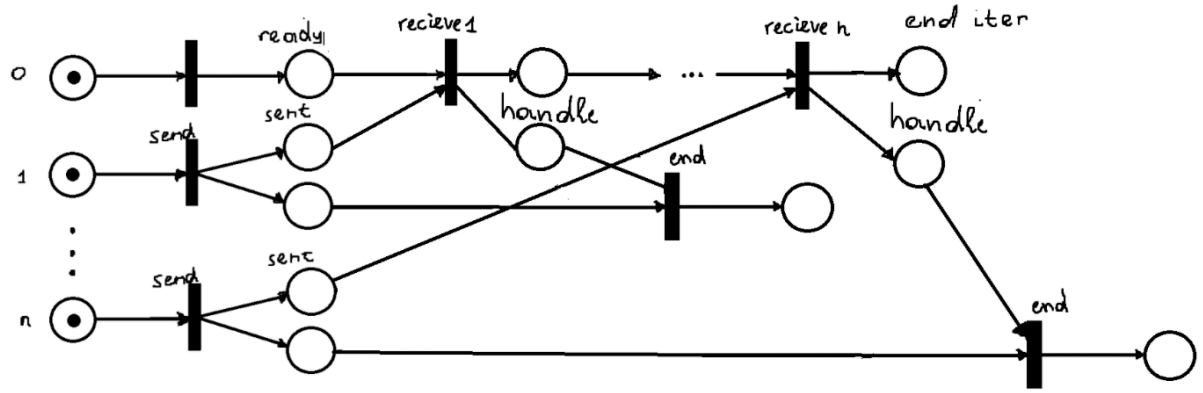

Рисунок 4 – Сеть Петри для измененной программы

Сравнивая две сети Петри, можно увидеть отличия в программах: в первом случае нулевой процесс ждет любого другого, а во втором ожидает их по порядковому номеру.

Вывод

MPI позволяет писать параллельные программы, где процессы могут взаимодействовать друг с другом через обмен сообщениями. Порядок выполнения операций в параллельных программах не всегда предсказуем и зависит от условий выполнения. В ходе работы был построен график зависимости времени от количества процессов и входных данных, который показывает, что чем больше количество процессов/размер входных данных, тем больше времени нужно на это. Также был построен график замедления, отражающий время выполнения параллельной программы бесконечно велико по сравнению с последовательной программой. Была модифицирована программа, с помощью которой можно последовательно вывести процессы. К обеим программам были нарисованы сети Петри, отражающие их основную логику.