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

ОТЧЕТ

по лабораторной работе №1

по дисциплине «Системы параллельной обработки данных»

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

Студент гр. 0303 _____ Калмак Д.А.

Преподаватель _____ Татаринов Ю.С.

Санкт-Петербург
2024

Цель работы.

Целью работы является написание и запуск параллельной программы MPI на разном количестве процессов, изучение зависимостей времени работы программы от числа процессов и построение сетей Петри.

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("\nHellofromprocess%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("\nHellofromprocess%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. С использованием языка программирования C++ написана параллельная программа MPI. Для работы с MPI была включена библиотека mpi.h. Для инициализации среды MPI используется MPI_Init. MPI_Comm_size позволяет получить общее количество процессов, а MPI_Comm_rank – ранг текущего процесса. Процесс с рангом 0 выводит сообщение о своем ранге и запускает цикл, где с помощью MPI_Recv получает сообщения с рангами от других процессов и выводит сообщение с рангом полученного процесса. Функция настроена на прием одного элемента типа MPI_INT в переменную RecvRank от процесса любого ранга и элементом любого тега, MPI_COMM_WORLD - коммуникатор, в рамках которого выполняется обмен сообщениями, а Status - структура данных с информацией о результате выполнения операции приема этих сообщений. Если процесс не с рангом 0, то он сообщение наоборот отсылает с помощью MPI_Send нулевому процессу. Функция настроена на отправку ранга процесса, который представляет собой один элемент типа MPI_INT с тегом 0, процессу с рангом 0, а MPI_COMM_WORLD – коммуникатор. MPI_Finalize завершает работу MPI, освобождая все ресурсы, связанные с ним. Программа представлена в листинге 1.

Листинг 1.

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

using namespace std;

int main(int argc, char** argv){
    int ProcNum, ProcRank, RecvRank;
    MPI_Status Status;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &ProcNum);
    MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);
```

```
    if (ProcRank == 0) {
        cout << "\n Hello from process " << ProcRank;
        for (int i = 1; i < ProcNum; i++) {
            MPI_Recv(&RecvRank,   1,   MPI_INT,   MPI_ANY_SOURCE,   MPI_ANY_TAG,
MPI_COMM_WORLD, &Status);
            cout << "\n Hello from process " << RecvRank;
        }
    } else {
        MPI_Send(&ProcRank, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
    }
    MPI_Finalize();

    return 0;
}
```

2-3. Программа была запущена на разных числах процессов: 1, 2, 4, 6, 8, 10, 12, 14, 16 – несколько раз. Каждый запуск первым выводом было сообщение с нулевым рангом, но затем порядок рангов в сообщениях разнился. Например, для четырех процессов были сообщения от рангов 0 2 1 3, а затем 0 3 1 2. Это вызвано тем, что процесс с нулевым рангом выбран главным, который пишет свое сообщение, а затем принимает сообщения с рангами других процессов, и в порядке получения выводит сообщения, а поскольку в функции приема выставлен аргумент MPI_ANY_SOURCE, который позволяет принимать сообщение от любого процесса, порядок процессов после нулевого недетерминированный. Для того, чтобы прием был упорядоченным, то есть порядок детерминированный, необходимо этот параметр поменять, что будет рассмотрено в седьмом пункте.

4. В программу были добавлены две проверки времени: начало и конец работы программы без учета ввода и вывода - с помощью функции MPI_Wtime, которая не меняет точку отсчета за время существования процесса. Изменный участок программы представлен в листинге 2.

Листинг 2.

```
if (ProcRank == 0) {
        start = MPI_Wtime();
        for (int i = 1; i < ProcNum; i++) {
            MPI_Recv(&RecvRank, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG,
MPI_COMM_WORLD, &Status);
        }
        end = MPI_Wtime();
```

```
        cout << "Time " << ProcNum << ": " << end - start << endl;
    }
```

Для 1, 4, 8, 12, 16 процессов было высчитано среднее время: 0.0000001, 0.00007824, 0.00019815, 0.00032262, 0.0004238417, - и построен график времени работы программы в зависимости от числа запущенных процессов, который представлен на рис. 1. Время выполнения программы представлено в листинге 3.

Листинг 3.

```
cut@cute:~/Документы/cpluscplus/3$ mpiexec.openmpi -n 1 --oversubscribe ./main
Time 1: 1e-07
cut@cute:~/Документы/cpluscplus/3$ mpiexec.openmpi -n 1 --oversubscribe ./main
Time 1: 1.5e-07
cut@cute:~/Документы/cpluscplus/3$ mpiexec.openmpi -n 1 --oversubscribe ./main
Time 1: 1.5e-07
cut@cute:~/Документы/cpluscplus/3$ mpiexec.openmpi -n 4 --oversubscribe ./main
Time 4: 0.00014335
cut@cute:~/Документы/cpluscplus/3$ mpiexec.openmpi -n 4 --oversubscribe ./main
Time 4: 6.3179e-05
cut@cute:~/Документы/cpluscplus/3$ mpiexec.openmpi -n 4 --oversubscribe ./main
Time 4: 2.8203e-05
cut@cute:~/Документы/cpluscplus/3$ mpiexec.openmpi -n 8 --oversubscribe ./main
Time 8: 0.000215295
cut@cute:~/Документы/cpluscplus/3$ mpiexec.openmpi -n 8 --oversubscribe ./main
Time 8: 0.00021252
cut@cute:~/Документы/cpluscplus/3$ mpiexec.openmpi -n 8 --oversubscribe ./main
Time 8: 0.000166648
cut@cute:~/Документы/cpluscplus/3$ mpiexec.openmpi -n 12 --oversubscribe ./main
Time 12: 0.000346438
cut@cute:~/Документы/cpluscplus/3$ mpiexec.openmpi -n 12 --oversubscribe ./main
Time 12: 0.000381748
cut@cute:~/Документы/cpluscplus/3$ mpiexec.openmpi -n 12 --oversubscribe ./main
Time 12: 0.000239673
cut@cute:~/Документы/cpluscplus/3$ mpiexec.openmpi -n 16 --oversubscribe ./main
Time 16: 0.000459235
cut@cute:~/Документы/cpluscplus/3$ mpiexec.openmpi -n 16 --oversubscribe ./main
Time 16: 0.000433877
cut@cute:~/Документы/cpluscplus/3$ mpiexec.openmpi -n 16 --oversubscribe ./main
Time 16: 0.000378413
```

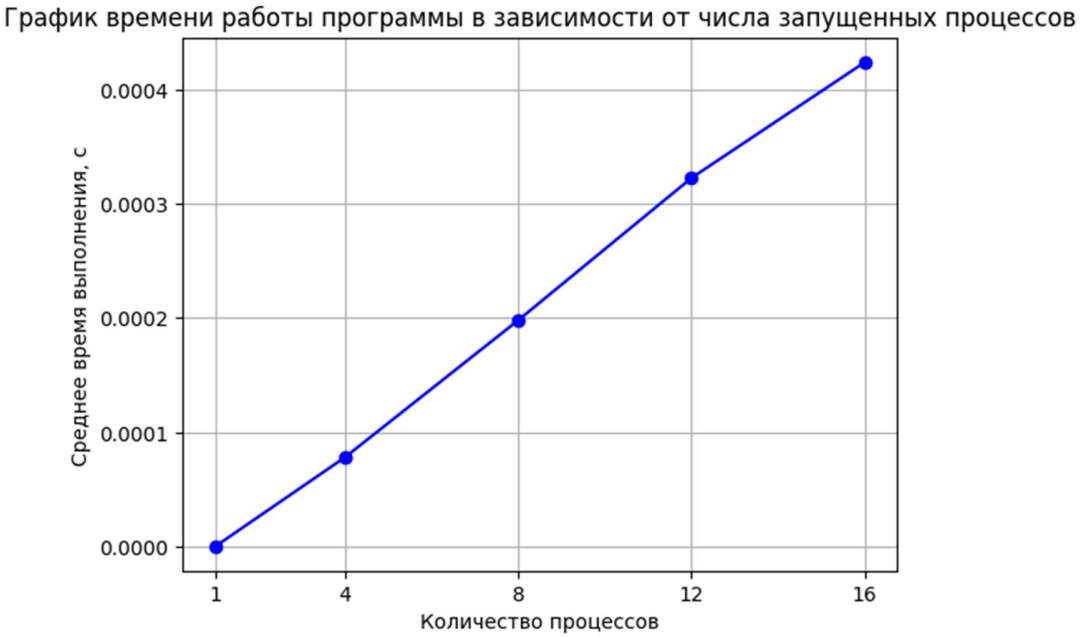

Рисунок 1 – График времени работы программы в зависимости от числа запущенных процессов

В случае, когда процесс один, выводится сообщение о его ранге и программа завершается. С увеличением числа процессов нулевой процесс должен не только вывести свое сообщение, но и принимать сообщения с рангами от других, что увеличивает количество принимаемых данных, и время выполнения программы увеличивается. Время выполнения программы может разниться при одинаковом числе процессов, что может быть вызвано как разной скоростью выполнения процессов, обработки, а также разным состоянием вычислительной машины.

5. Для построения графика ускорения/замедления работы программы необходимо разделить время выполнения программы с 1 процессом на время выполнения программы с 4, 8, 12, 16 процессами соотвественно. Поскольку разница во времени между 1 процессом и 4, 8, 12, 16 процессами большая, было построено два графика: с 1, 4, 8, 12, 16 процессами и с 4, 8, 12, 16, чтобы увеличить часть, где заметна разница в замедлении. Графики замедления представлены на рис. 2-3.

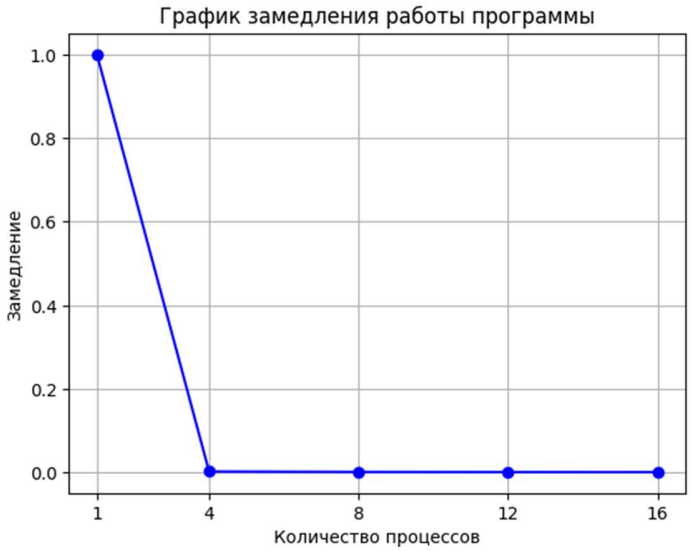

Рисунок 2 – График замедления работы программы (1, 4, 8, 12, 16)

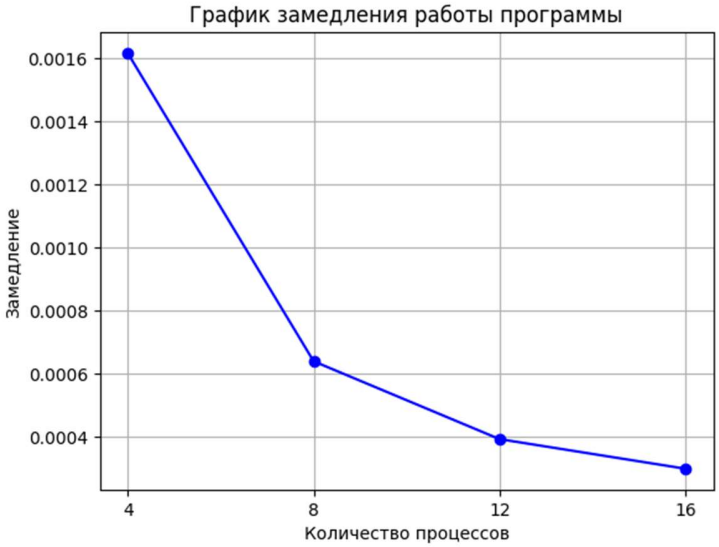

Рисунок 3 – График замедления работы программы (4, 8, 12, 16)

6. Чтобы порядок вывода сообщений на экран соответствовал номеру соответствующего процесса необходимо модифицировать функцию MPI_Recv. В ней есть параметр source, отвечающий за ранг процесса, от которого нужно выполнить прием. Поскольку у нас есть цикл for, можно использовать счетчик цикла как параметр source, то есть ранг процесса на прием, и в цикле процессы будут обрабатываться последовательно. Модифицированный участок кода представлен в листинге 4.

Листинг 4.

```
if (ProcRank == 0) {
        cout << "\n Hello from process " << ProcRank;
        for (int i = 1; i < ProcNum; i++) {
            MPI_Recv(&RecvRank,  1,  MPI_INT,  i,  MPI_ANY_TAG,  MPI_COMM_WORLD,
&Status);
            cout << "\n Hello from process " << RecvRank;
        }
```

7. Построены две сети Петри. Первая сеть Петри представлена на рис. 4 и отражает параллельную программу для трех процессов, соответствующую первому заданию, где вывод ранга процессов был недетерминированный. Вторая сеть Петри представлена на рис. 5 и отражает параллельную программу для трех процессов, модифицированную для детерминированного вывода ранга процессов.

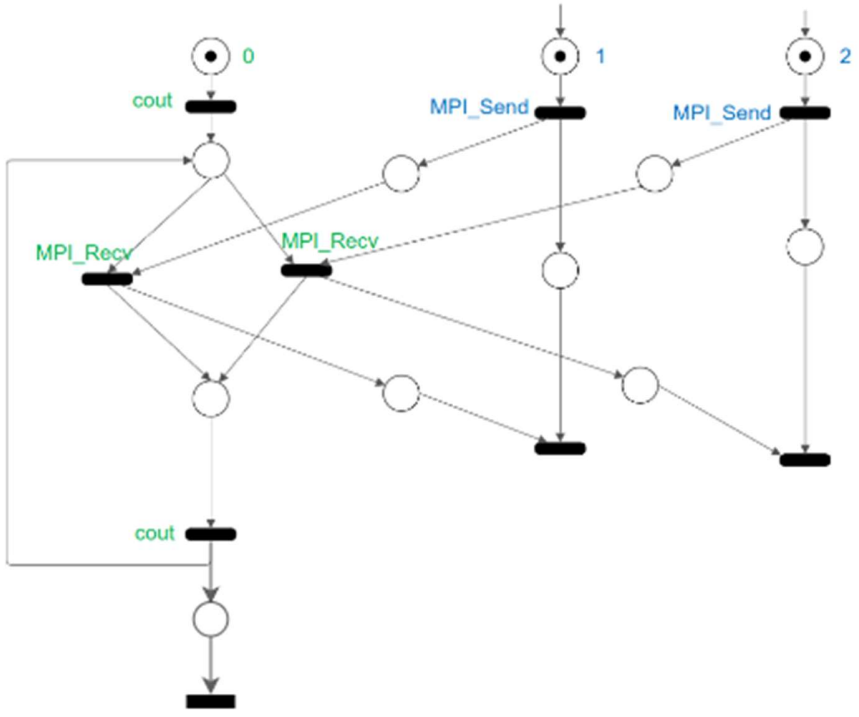

Рисунок 4 – Сеть Петри для параллельной программы с недетерменированным выводом ранга процессов для трех процессов

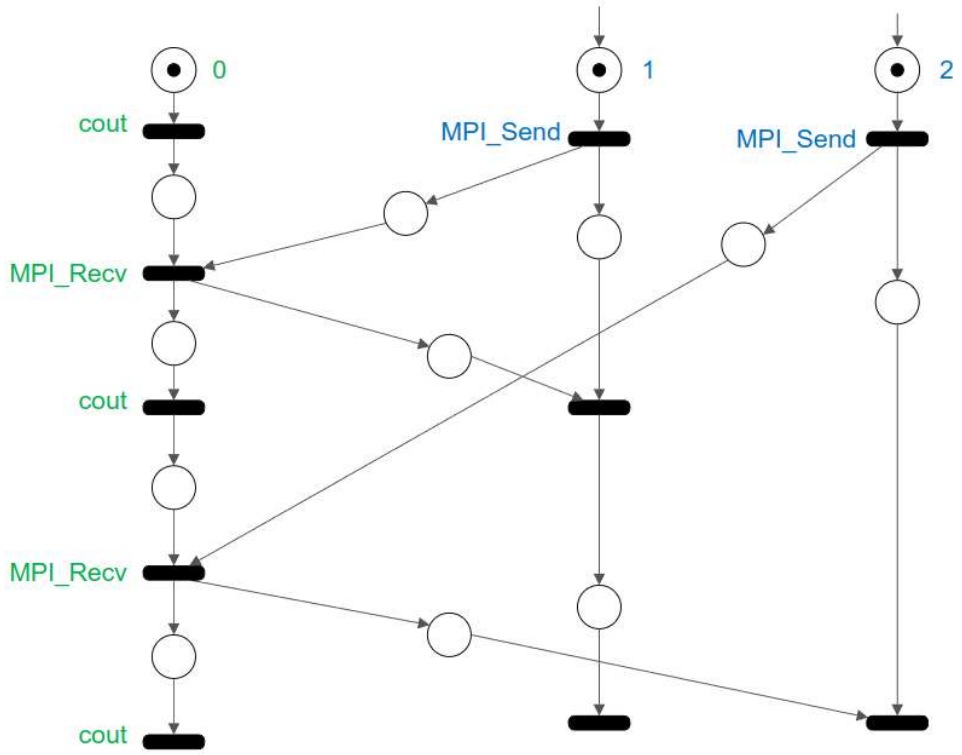

Рисунок 5 – Сеть Петри для параллельной программы с детерменированным выводом ранга процессов для трех процессов

Вывод.

Таким образом, было освоено написание параллельных программ MPI и их запуска.

В результате выполнения первой программы вывод ранга процессов был неупорядоченным и порядок был разным от запуска к запуску, но нулевой процесс всегда выводился в начале, он был выбран для приема сообщений о рангах других процессов.

Были изучены зависимости времени работы программы от числа процессов. С увеличением числа процессов количество данных, которые нужно обработать нулевому процессу, возрастает, и время работы программы увеличивается. Также время выполнения программы при одинаковом количестве процессов было разным, что может быть вызвано разной скоростью

выполнения процессов, обработки, разным состоянием вычислительной машины. На основании полученных данных о времени работы программы построены график времени работы программы в зависимости от числа запущенных процессов и графики замедления работы программы.

Программа была модифицирована для детерминированного вывода. Для программы с неупорядоченным выводом и модифицированной программы были построены сети Петри.