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

ОТЧЕТ

по лабораторной работе №2
по дисциплине «Системы параллельной обработки данных»
Тема: Использование функций обмена данными «точка-точка» в библиотеке MPI

Студент гр. 0303 _____ Калмак Д.А.

Преподаватель _____ Татаринов Ю.С.

Санкт-Петербург
2024

Цель работы.

Целью работы является написание параллельной программы MPI с обменом данными «точка-точка», изучение зависимостей времени работы программы от количества данных и числа процессов и построение сети Петри.

Задание.

1. Написать программу «Суммирование элементов массива». Процесс 0 генерирует массив и раздает его другим процессам для вычисления локальных сумм, после чего вычисляет общую сумму.
2. Краткое описание выбранного алгоритма решения задачи и листинг программы.
3. Нарисовать сеть Петри для MPI программы.
4. Распечатка результатов и времени работы программы на разном количестве процессов и для различных объемов исходных данных.
5. Построить графики зависимости времени выполнения программы, как от числа запущенных процессов, так и от размерности решаемой задачи, т.е. при разных объемах исходных данных.
6. Построить графики ускорения/замедления программы.

Выполнение работы.

1-2. С использованием языка программирования C++ написана параллельная программа MPI с использованием функций обмена данными «точка-точка» для суммирования элементов массива. 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. Переменная N задает количество элементов в массиве, а array – вектор для чисел. Переменная local_sum – для локальных сум, а general_sum – общая сумма. Если процесс нулевой, то вектор

array заполняется числами. Для распределения каждому процессу нужного числа элементов используется три переменных: quotient – это частное, которое хранит в себе базовое число элементов для каждого процесса, remainder хранит остаток количества элементов, которые нужно распределить по процессам, и local_N – конечное число элементов для каждого процесса. Для хранения локальных элементов каждый процесс имеет вектор local_array. Процесс с нулевым рангом отправляет остальным процессам элементы с помощью функции MPI_Send. Число элементов у каждого процесса индивидуальное, поэтому offset и send_count позволяют контролировать указатель и индивидуальное количество элементов, передаваемые в функцию. Иначе если процесс ненулевого ранга, то с помощью функции MPI_Recv он получает вектор элементов для вычисления локальной суммы. Процесс ненулевого ранга рассчитывает локальную сумму и с помощью функции MPI_Send отправляет ее нулевому процессу. А процесс нулевого ранга получает локальные суммы с процессов с помощью функции MPI_Recv в переменную received_sum и суммирует с general_sum, что после получения локальных сумм со всех процессов имеет сумму всех элементов массива. Программа представлена в листинге 1.

Листинг 1.

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

using namespace std;

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

    int N = 100000;
    vector<int> array(N);

    int local_sum = 0;
```

```
    int general_sum = 0;

    if (ProcRank == 0) {
        for (size_t i = 0; i < N; i++) {
            array[i] = 1;
        }
    }

    int quotient = N / (ProcNum - 1);
    int remainder = N % (ProcNum - 1);
    int local_N = quotient + (ProcRank - 1 < remainder ? 1 : 0);

    vector<int> local_array(local_N);

    double start, end;

    if (ProcRank == 0) {
        int offset = 0;
        start = MPI_Wtime();
        for (int i = 1; i < ProcNum; i++) {
            int send_count = quotient + (i - 1 < remainder ? 1 : 0);

            MPI_Send(&array[offset], send_count, MPI_INT, i, 0, MPI_COMM_WORLD);
            offset += send_count;
        }
    } else {
        MPI_Recv(local_array.data(),  local_N,  MPI_INT,  0,  0,  MPI_COMM_WORLD,
&Status);
    }

    if (ProcRank != 0) {
        for (size_t i = 0; i < local_N; i++) {
            local_sum += local_array[i];
        }

        MPI_Send(&local_sum, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
    } else {
        general_sum = 0;
        for (int i = 1; i < ProcNum; i++) {
            int received_sum;
            MPI_Recv(&received_sum, 1, MPI_INT, i, 0, MPI_COMM_WORLD, &Status);
            general_sum += received_sum;
        }
        end = MPI_Wtime();
    }

    if (ProcRank == 0) {
        cout << "Общая сумма: " << general_sum << endl;
        cout << "Time " << end - start << endl;
    }

    MPI_Finalize();

    return 0;
}
```

3. Построена сеть Петри. Сеть Петри представлена на рис. 1 и отражает параллельную программу для трех процессов, в которой происходит вычисление суммы массива.

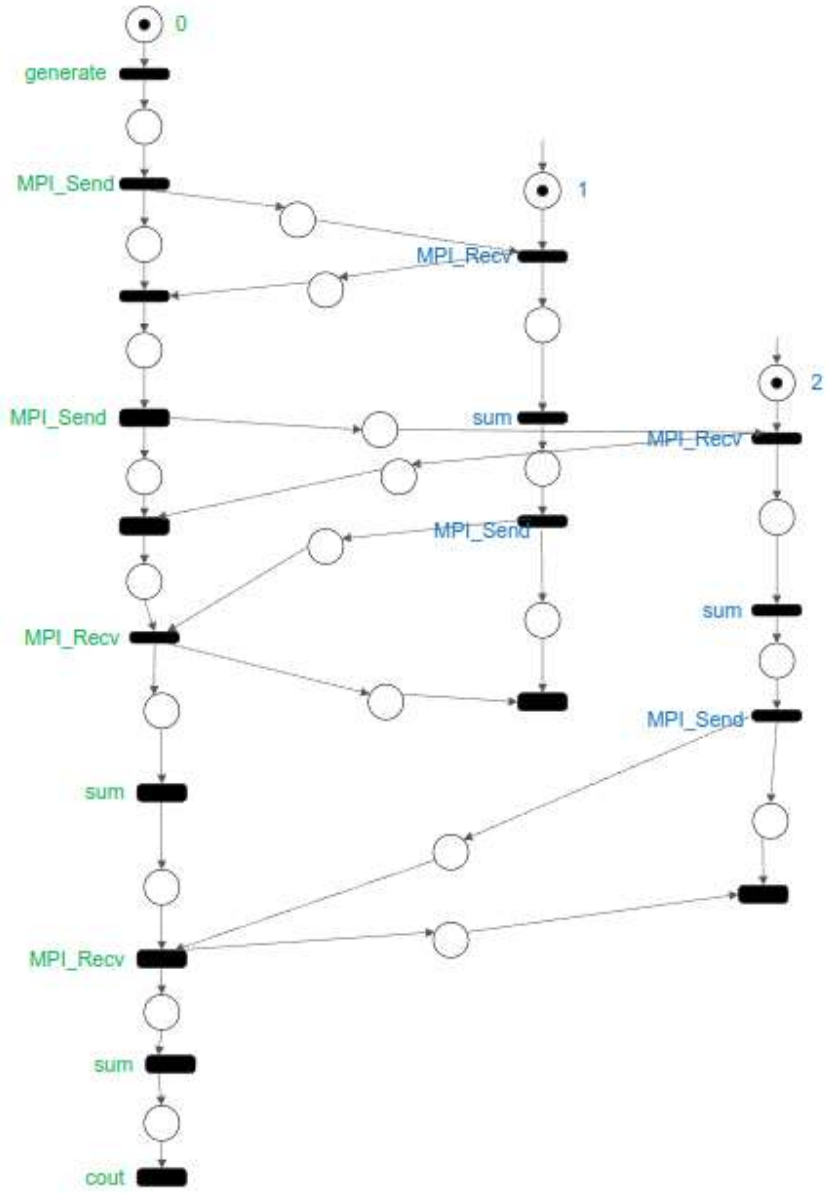

Рисунок 1 – Сеть Петри для параллельной программы, складывающей сумму элементов, для трех процессов

4. Программа была запущена на разных числах процессов: 2, 4, 6, 8, 10, 12, 14, 16. Результат работы программы, а именно сумма элементов массива, и время выполнения представлены в листинге 2.

Листинг 2.

```
cut@cute:~/Документы/cpluscplus/2$   mpicxx.openmpi   main.cpp   -o   ./main   &&
mpiexec.openmpi -n 2 --oversubscribe ./main
Общая сумма: 100000
Time 0.000366865
cut@cute:~/Документы/cpluscplus/2$   mpicxx.openmpi   main.cpp   -o   ./main   &&
mpiexec.openmpi -n 4 --oversubscribe ./main
Общая сумма: 100000
Time 0.000382624
cut@cute:~/Документы/cpluscplus/2$   mpicxx.openmpi   main.cpp   -o   ./main   &&
mpiexec.openmpi -n 6 --oversubscribe ./main
Общая сумма: 100000
Time 0.000489395
cut@cute:~/Документы/cpluscplus/2$   mpicxx.openmpi   main.cpp   -o   ./main   &&
mpiexec.openmpi -n 8 --oversubscribe ./main
Общая сумма: 100000
Time 0.000681985
cut@cute:~/Документы/cpluscplus/2$   mpicxx.openmpi   main.cpp   -o   ./main   &&
mpiexec.openmpi -n 10 --oversubscribe ./main
Общая сумма: 100000
Time 0.00108912
cut@cute:~/Документы/cpluscplus/2$   mpicxx.openmpi   main.cpp   -o   ./main   &&
mpiexec.openmpi -n 12 --oversubscribe ./main
Общая сумма: 100000
Time 0.0028997
cut@cute:~/Документы/cpluscplus/2$   mpicxx.openmpi   main.cpp   -o   ./main   &&
mpiexec.openmpi -n 14 --oversubscribe ./main
Общая сумма: 100000
Time 0.00327107
cut@cute:~/Документы/cpluscplus/2$   mpicxx.openmpi   main.cpp   -o   ./main   &&
mpiexec.openmpi -n 16 --oversubscribe ./main
Общая сумма: 100000
Time 0.00355325
```

Программа была также запущена с разным количеством исходных данных, а именно количеством элементов в массиве: 100000, 200000, 300000, 400000, 500000 - на четырех процессах. Результат работы программы, а именно сумма элементов массива, и время выполнения представлены в листинге 3.

Листинг 3.

```
cut@cute:~/Документы/cpluscplus/2$   mpicxx.openmpi   main.cpp   -o   ./main   &&
mpiexec.openmpi -n 4 —oversubscribe ./main
Общая сумма: 100000
Time 0.000346379
```

```
cut@cute:~/Документы/cpluscplus/2$   mpicxx.openmpi   main.cpp   -o   ./main   &&
mpiexec.openmpi -n 4 —oversubscribe ./main
Общая сумма: 200000
Time 0.000552981
cut@cute:~/Документы/cpluscplus/2$   mpicxx.openmpi   main.cpp   -o   ./main   &&
mpiexec.openmpi -n 4 —oversubscribe ./main
Общая сумма: 300000
Time 0.000737852
cut@cute:~/Документы/cpluscplus/2$   mpicxx.openmpi   main.cpp   -o   ./main   &&
mpiexec.openmpi -n 4 —oversubscribe ./main
Общая сумма: 400000
Time 0.000873994
cut@cute:~/Документы/cpluscplus/2$   mpicxx.openmpi   main.cpp   -o   ./main   &&
mpiexec.openmpi -n 4 —oversubscribe ./main
Общая сумма: 500000
Time 0.00123247
```

5. На основе полученных данных при выполнении программы на разных количествах процессов построен график зависимости времени выполнения программы от числа запущенных процессов, который представлен на рис. 2. Также построен график зависимости времени выполнения программы от количества данных на основе выполнения работы программы с разным числом элементов и представлен на рис. 3.

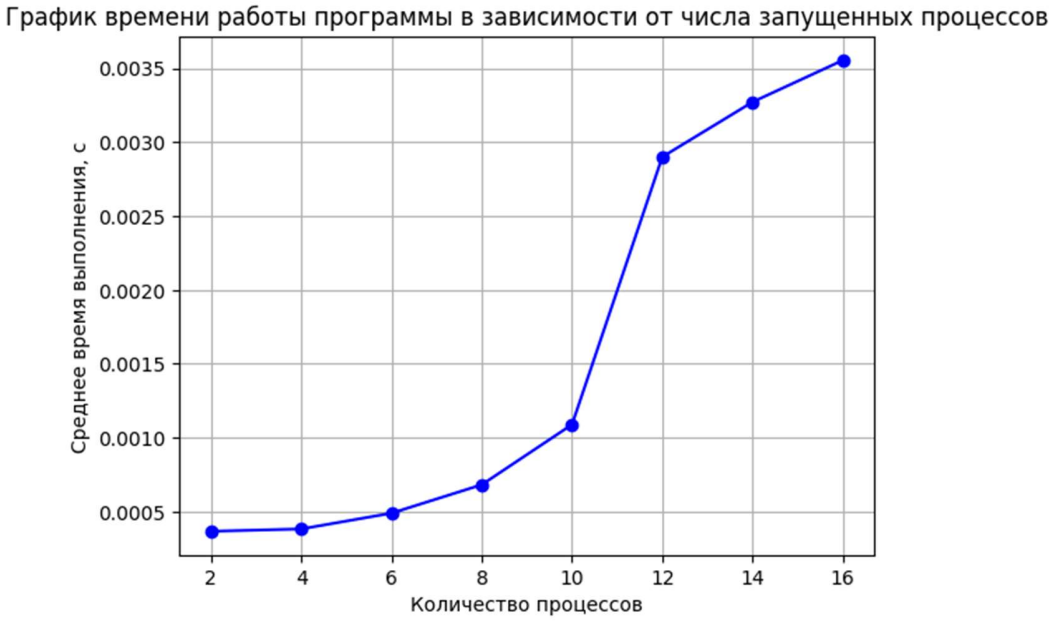

Рисунок 2 – График времени работы программы в зависимости от числа запущенных процессов

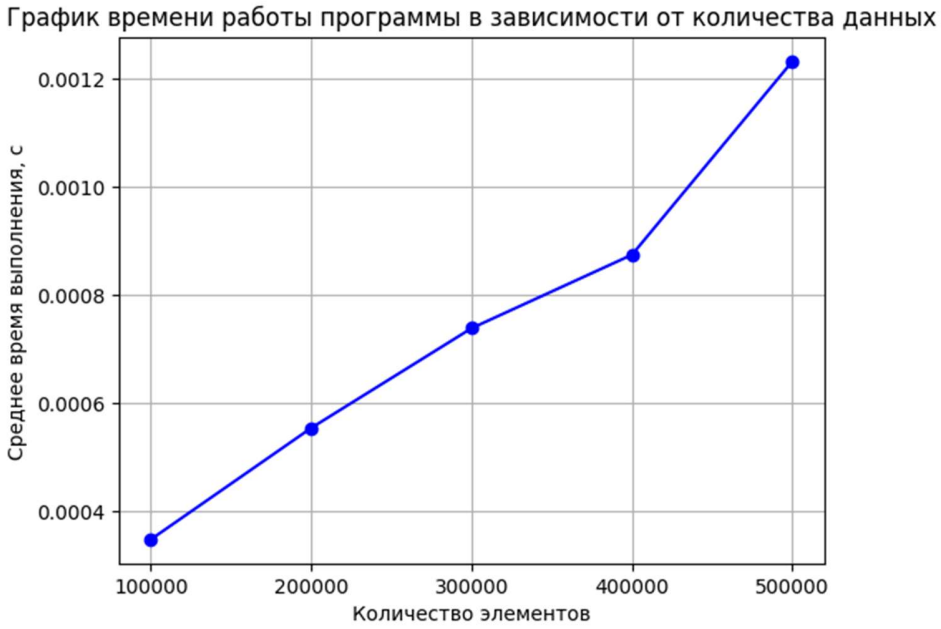

Рисунок 3 – График времени работы программы в зависимости от количества данных

Исходя из графика, представленного на рис. 2, время работы программы увеличивается с ростом количества запущенных процессов. Процесс с рангом ноль является узким звеном, поскольку ему нужно распределить элементы массива по остальным процессам, а потом получить со всех процессов вычисленные локальные суммы, а затем сложить их, что с ростом числа процессов приводит к увеличению производимых действий нулевым процессом с передачей «точка-точка» и соотвественно ведет к увеличению времени работы программы. С увеличением количества элементов в массиве, для которого нужно вычислить сумму элементов, исходя из графика, представленного на рис. 3, возрастает время работы программы, что связано с большим количеством производимых операций и объемом данных. Время выполнения программы может разниться при одинаковых условиях, что может быть вызвано как разной скоростью выполнения процессов, обработки, а также разным состоянием вычислительной машины.

6. Для построения графика ускорения/замедления работы программы с разным количеством запущенных процессов необходимо разделить время выполнения программы с 2 процессами на время выполнения программы с 4, 6, 8, 10, 12, 14, 16 процессами соотвественно. График замедления представлен на рис. 4.

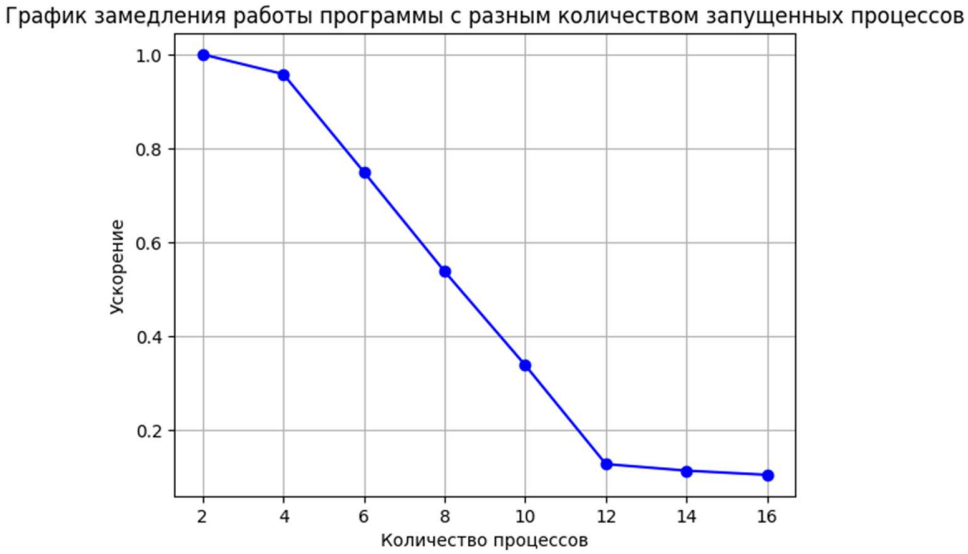

Рисунок 4 – График замедления работы программы с разным количеством запущенных процессов

Для построения графика ускорения/замедления работы программы с разным количеством элементов необходимо разделить время выполнения программы с 100000 элементами на время выполнения программы с 200000, 300000, 400000, 500000 элементами соотвественно. График замедления представлен на рис. 5.

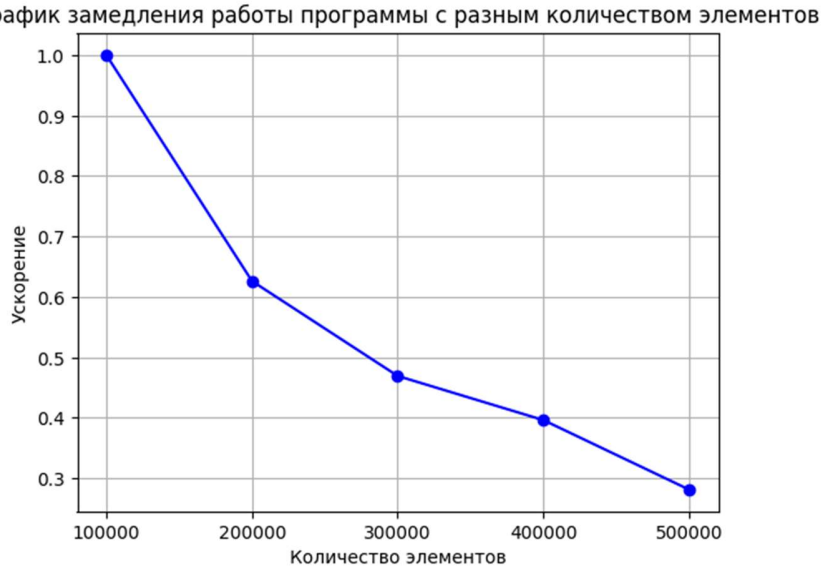

Рисунок 5 – График замедления работы программы с разным количеством элементов

Вывод.

Таким образом, было освоено написание параллельных программ MPI с использованием функций обмена данными «точка-точка» в библиотеке MPI.

Были изучены зависимости времени работы программы от числа запущенных процессов и количества элементов. С увеличением числа процессов время работы программы увеличивается, поскольку нулевой процесс имеет больше производимых действий и осуществляет передачу «точка-точка». Также время выполнения программы при увеличении числа элементов в массиве возрастает, что связано с увеличением числа операций и объема данных. На основании полученных данных о времени работы программы построены графики времени работы программы в зависимости от числа запущенных процессов и числа элементов и соответствующие им графики замедления работы программы. Для программы была построена сеть Петри.