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

ОТЧЕТ

по лабораторной работе №3
по дисциплине «Системы параллельной обработки данных»
Тема: Использование аргументов-джокеров

Студент гр. 0303 _____ Калмак Д.А.

Преподаватель _____ Татаринов Ю.С.

Санкт-Петербург
2024

Цель работы.

Целью работы является написание параллельной программы MPI с аргументами-джокерами, изучение зависимостей времени работы программы от количества данных и числа процессов и построение сети Петри.

Задание.

1. Написать программу «Имитация посылки пакета с маршрутом». Процесс 0 генерирует сообщение, которое состоит из маршрутной и информационной части. Маршрутная часть содержит последовательность номеров процессов – промежуточных адресатов. Промежуточный адресат, получив пакет, убирает из адресной части свой номер и передает сообщение дальше согласно списку. Окончательный адресат, получив посылку, отчитывается перед процессом 0.
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 аргументов-джокеров для посылки пакета с маршрутом. 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. Чтобы процесс генерации был более случайным, причем даже на маленьком числе процессов, для генератора случайных чисел используются: std::random_device rd, чтобы получить случайное число, которое обеспечивает более высокую степень случайности и используется для инициализации std::mt19937 gen(rd()), что является генератором псевдослучайных чисел на основе алгоритма Mersenne Twister. Нулевой процесс генерирует случайный маршрут с помощью функции generateRandomRoute(vector<int>& route, int procNum), которая принимает на вход аргументы в виде ссылки на вектор route, который будет хранить случайный сгенерированный маршрут, и ProcNum, который содержит количество запущенных процессов. Изначально route заполняем числами с 1 по ProcNum - 1, поскольку нулевой процесс не входит в маршрутную часть сообщения. Затем используем функцию shuffle, которая перемешивает вектор route и в качестве генератора используется gen. После генерации маршрута вектор приводится к строке, где каждый адресат отделяется символом “|”, тем самым создаётся маршрутная часть сообщения. Нулевой процесс также генерирует информационную часть, используя функцию generateRandomInfoPart(string& infoPart, int infoPartLen), которая принимает на вход аргументы в виде ссылки на строку infoPart, которая будет хранить случайную информационную часть, и infoPartLen, который задает длину строки. Для генератора букв строки также используется gen. После генерации маршрутная часть совмещается с информационной частью, тем самым образуется конечное сообщение, которое передается первому адресату с помощью функции MPI_Send. Ненулевой процесс, или адресат, получает с помощью функции MPI_Recv сообщение. Заданы джокеры MPI_ANY_SOURCE, MPI_ANY_TAG. Процесс удаляет себя из маршрутной части. С помощью функции find производится поиск “|”, чтобы определить был

ли текущий адресат промежуточным или конечным, для этого результат выполнения функции сравнивается с string::npos. Если адресат был промежуточным, то определяется следующий адресат от начала строки до значения функции find, и с помощью MPI_Send ему отправляется сообщение. Аналогичные действия производятся до тех пор, пока адресат не будет конечным. В случае конечного адресата процесс отправляет свой ранг нулевому процессу, сообщая, что он, как конечный адресат получил сообщение. Процесс с рангом 0 получает отчет от последнего адресата с помощью функции MPI_Recv, в которой заданы джокеры MPI_ANY_SOURCE, MPI_ANY_TAG. Программа представлена в листинге 1. Пример работы программы с логированием представлен в листинге 2.

Листинг 1.

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

using namespace std;

std::random_device rd;
std::mt19937 gen(rd());

void generateRandomRoute(vector<int>& route, int ProcNum) {
    for (int i = 1; i < ProcNum; i++) {
        route.push_back(i);
    }
    std::shuffle(route.begin(), route.end(), gen);
}

void generateRandomInfoPart(string& infoPart, int infoPartLen) {
    for (int i = 0; i < infoPartLen; i++) {
        char letter = 'a' + (gen() % 26);
        infoPart += letter;
    }
}

int main(int argc, char** argv) {
    MPI_Init(&argc, &argv);

    int ProcNum, ProcRank;
    MPI_Comm_size(MPI_COMM_WORLD, &ProcNum);
```

```
    MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);

    double start, end;

    if (ProcRank == 0) {
        vector<int> route;
        generateRandomRoute(route, ProcNum);
        string routePart;
        for (int r : route) {
            routePart += to_string(r) + "|";
        }
        // string infoPart = "Hello, this is the part with information.";
        string infoPart;
        generateRandomInfoPart(infoPart, 100);
        string fullMessage = routePart + infoPart;

        // cout << "Process 0 and message '" << fullMessage << "'" << endl;

        start = MPI_Wtime();
        MPI_Send(fullMessage.c_str(),   fullMessage.size()   +   1,   MPI_CHAR,
route[0], 0, MPI_COMM_WORLD);
        // cout << "Process " << ProcRank << " sent message to Process " <<
route[0] << endl;
    } else {
        char msg[200];
        MPI_Recv(msg,   sizeof(msg),   MPI_CHAR,   MPI_ANY_SOURCE,   MPI_ANY_TAG,
MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        // cout << "Process " << ProcRank << " received message: '" << msg <<
"'" << endl;

        string message(msg);
        message.erase(message.begin(), message.begin() + message.find('|') + 1);
        int pos = message.find('|');

        if (pos != string::npos) {
            int nextRoute = stoi(message.substr(0, pos));

            MPI_Send(message.c_str(),  message.size()  +  1,  MPI_CHAR,  nextRoute,
0, MPI_COMM_WORLD);
            // cout << "Process " << ProcRank << " forwarded message to Process
" << nextRoute << endl;
        } else {
            // cout << "Process " << ProcRank << " was final and received the
message: " << message << endl;
            MPI_Send(&ProcRank, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
            // cout << "Process " << ProcRank << " reported to Process 0." <<
endl;
        }
    }

    if (ProcRank == 0) {
        int FinalRank;
        MPI_Recv(&FinalRank,    1,    MPI_INT,    MPI_ANY_SOURCE,    MPI_ANY_TAG,
MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        end = MPI_Wtime();
        // cout << "Final addressee, Process " << FinalRank << ", received the
message and reported to me, Process 0." << endl;
        cout << "Time " << end - start << endl;
```

```
    }

    MPI_Finalize();
    return 0;
}
```

Листинг 2.

```
cut@cute:~/Документы/cpluscplus/3$ mpiexec.openmpi -n 7 --oversubscribe ./main
Process 0 and message '1|4|2|6|3|5|lzufabkoej'
Process 0 sent message to Process 1
Process 1 received message: '1|4|2|6|3|5|lzufabkoej'
Process 1 forwarded message to Process 4
Process 4 received message: '4|2|6|3|5|lzufabkoej'
Process 4 forwarded message to Process 2
Process 2 received message: '2|6|3|5|lzufabkoej'
Process 2 forwarded message to Process 6
Process 6 received message: '6|3|5|lzufabkoej'
Process 6 forwarded message to Process 3
Process 3 received message: '3|5|lzufabkoej'
Process 3 forwarded message to Process 5
Process 5 received message: '5|lzufabkoej'
Process 5 was final and received the message: lzufabkoej
Process 5 reported to Process 0.
Final addressee, Process 5, received the message and reported to me, Process 0.
Time 0.000432783
```

3. Построена сеть Петри. Сеть Петри представлена на рис. 1 и отражает параллельную программу для трех процессов, в которой происходит имитация посылки пакета с маршрутом 1-2.

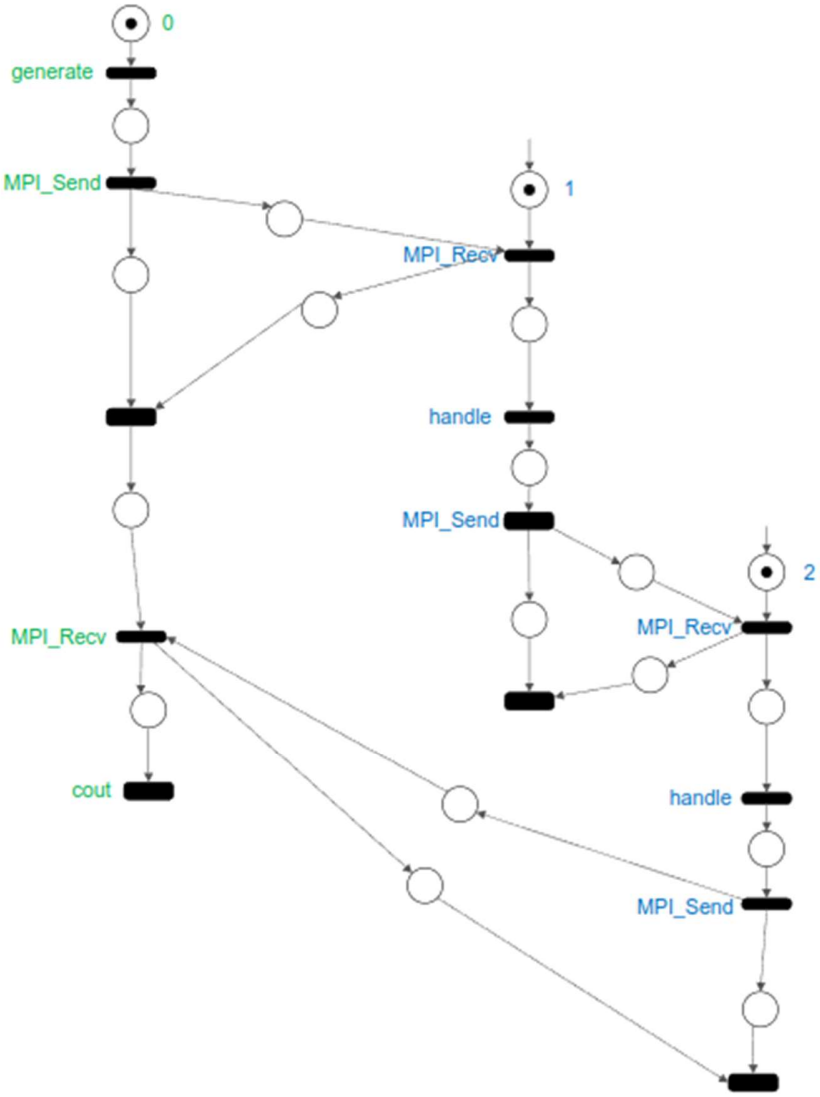

Рисунок 1 – Сеть Петри для параллельной программы, имитирующей посылку пакета с маршрутом 1-2, для трех процессов

4. Программа была запущена на разных числах процессов: 2, 4, 6, 8, 10, 12, 14, 16. Время выполнения программы с посылкой пакета в зависимости от количества процессов представлено в листинге 2.

Листинг 2.

```
cut@cute:~/Документы/cpluscplus/3$ mpiexec.openmpi -n 2 --oversubscribe ./main
Time 9.695e-05
cut@cute:~/Документы/cpluscplus/3$ mpiexec.openmpi -n 4 --oversubscribe ./main
Time 0.000132998
```

```
cut@cute:~/Документы/cpluscplus/3$ mpiexec.openmpi -n 6 --oversubscribe ./main
Time 0.000358277
cut@cute:~/Документы/cpluscplus/3$ mpiexec.openmpi -n 8 --oversubscribe ./main
Time 0.000419061
cut@cute:~/Документы/cpluscplus/3$ mpiexec.openmpi -n 10 --oversubscribe ./main
Time 0.000550605
cut@cute:~/Документы/cpluscplus/3$ mpiexec.openmpi -n 12 --oversubscribe ./main
Time 0.000635554
cut@cute:~/Документы/cpluscplus/3$ mpiexec.openmpi -n 14 --oversubscribe ./main
Time 0.00113989
cut@cute:~/Документы/cpluscplus/3$ mpiexec.openmpi -n 16 --oversubscribe ./main
Time 0.00179784
```

Программа была также запущена с разным количеством исходных данных, а именно были сгенерированы информационные части сообщения с 100000, 200000, 300000, 400000, 500000 элементами, на четырех процессах. Время выполнения программы с посылкой пакета в зависимости от объема данных в нем представлено в листинге 3.

Листинг 3.

```
cut@cute:~/Документы/cpluscplus/3$ mpicxx.openmpi main.cpp -o ./main && mpiexec.openmpi -n 4 --oversubscribe ./main
Time 0.000581159
cut@cute:~/Документы/cpluscplus/3$ mpicxx.openmpi main.cpp -o ./main && mpiexec.openmpi -n 4 --oversubscribe ./main
Time 0.000744987
cut@cute:~/Документы/cpluscplus/3$ mpicxx.openmpi main.cpp -o ./main && mpiexec.openmpi -n 4 --oversubscribe ./main
Time 0.000998352
cut@cute:~/Документы/cpluscplus/3$ mpicxx.openmpi main.cpp -o ./main && mpiexec.openmpi -n 4 --oversubscribe ./main
Time 0.0020341
cut@cute:~/Документы/cpluscplus/3$ mpicxx.openmpi main.cpp -o ./main && mpiexec.openmpi -n 4 --oversubscribe ./main
Time 0.00220531
```

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 размером сообщения и представлен на рис. 3.

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

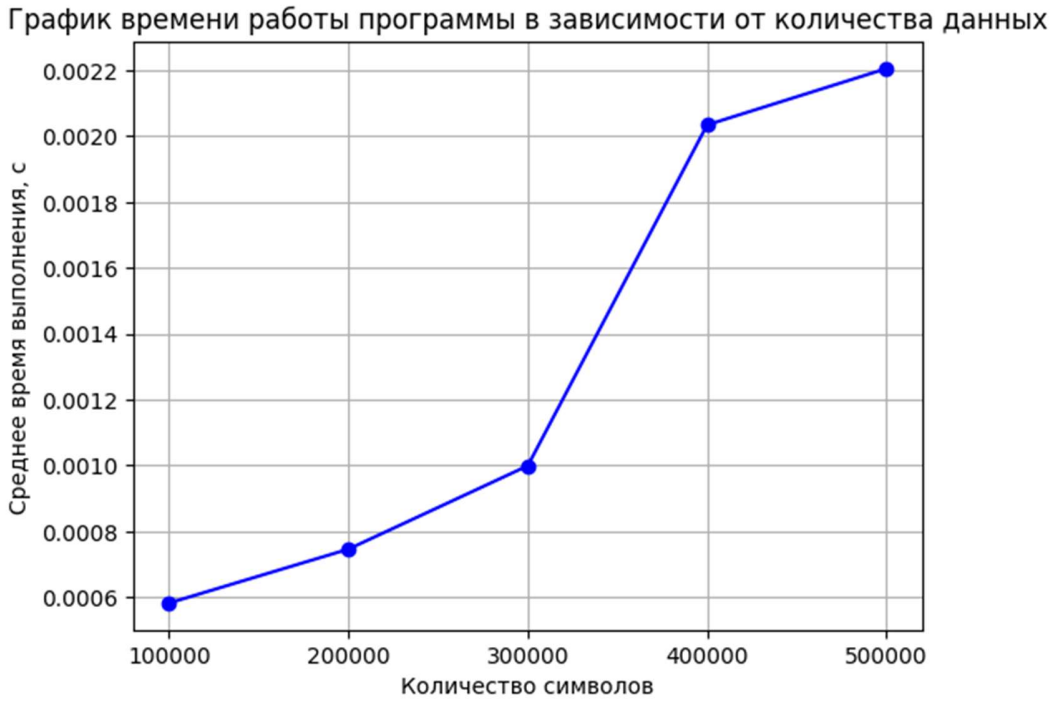

Рисунок 3 – График времени работы программы в зависимости от количества данных

Исходя из графика, представленного на рис. 2, время работы программы увеличивается с ростом количества запущенных процессов. Увеличение количества процессов означает увеличение количества промежуточных адресатов, то есть пакет проходит больший путь, задействует больше процессов и требует больше времени на обработку маршрутной части. С увеличением количества данных в сообщении, исходя из графика, представленного на рис. 3, возрастает время работы программы, что связано с затратами на передачу информации большего объема. Время выполнения программы может быть разным при одинаковых условиях, что может быть вызвано как разной скоростью выполнения процессов, обработки данных и их проверки, 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.

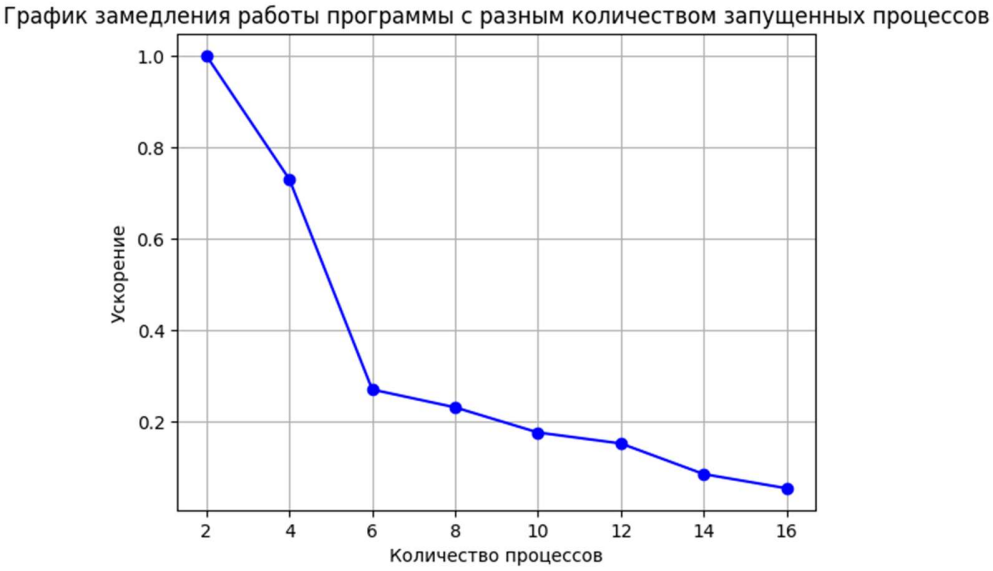

Рисунок 4 – График замедления работы программы с разным количеством запущенных процессов

Для построения графика ускорения/замедления работы программы с разным размером сообщения необходимо разделить время выполнения программы с 100000 символами в информационной части на время выполнения программы с 200000, 300000, 400000, 500000 символами соотвественно. График замедления представлен на рис. 5.

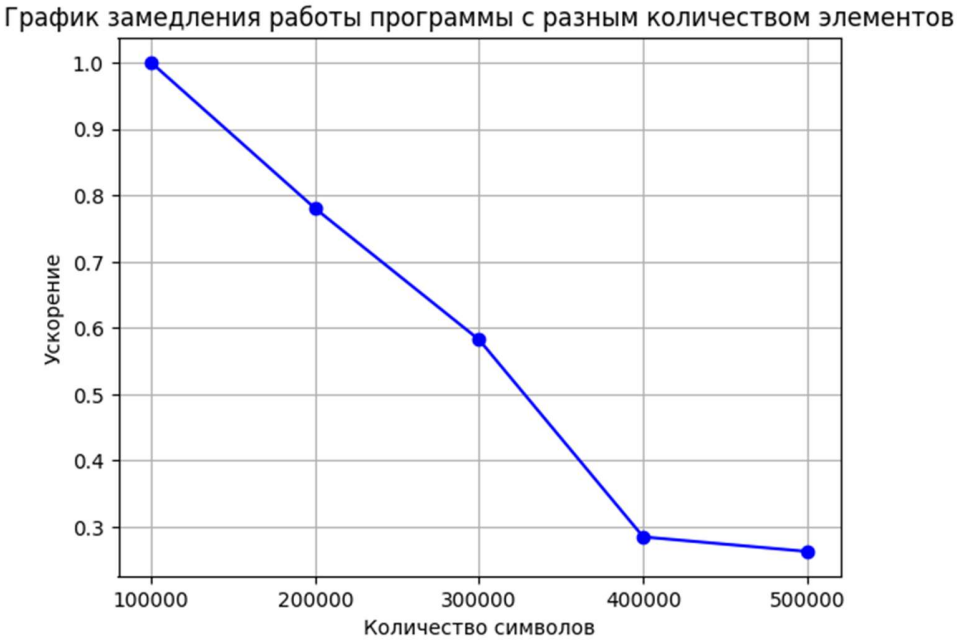

Рисунок 5 – График замедления работы программы с разным количеством элементов

Вывод.

Таким образом, было освоено написание параллельных программ MPI с использованием аргументов-джокеров библиотеки MPI и имитацией посылки пакета по маршруту.

Были изучены зависимости времени работы программы от числа запущенных процессов и размера сообщения. С увеличением числа процессов время работы программы увеличивается, поскольку маршрут увеличивается, тем самым требуется больше времени на посылку пакета по маршруту, а так же обработку маршрутной части сообщения. Также время выполнения программы возрастает при увеличении размера сообщения, что связано с увеличением

объема данных для передачи. На основании полученных данных о времени работы программы построены графики времени работы программы в зависимости от числа запущенных процессов и количества данных и соответствующие им графики замедления работы программы. Для программы была построена сеть Петри.