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

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Системы параллельной обработки данных»

Тема: Коллективные операции

Студент гр. 0303 _____ Калмак Д.А.

Преподаватель _____ Татаринов Ю.С.

Санкт-Петербург
2024

Цель работы.

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

Задание.

1. Написать параллельную программу вычисления суммы элементов вектора с использованием коллективных операций.
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. Переменная N отвечает за количество элементов в векторе. local_sum и general_sum локальные суммы процессов и общая сумма. Поскольку количество элементов может не соответствовать количеству процессов в равном количестве, вектор local_N предназначен для количества элементов

каждому процессу, a displs - для смещений для каждого процесса, чтобы правильно распределить данные. quotient – это частное, которое хранит в себе базовое число элементов для каждого процесса, remainder хранит остаток количества элементов, которые нужно распределить по процессам. Для хранения локальных элементов каждый процесс имеет вектор local_array. Вектор array заполняется в нулевом процессе. Для распределения элементов используется коллективная операция MPI_Scatterv, причем используется MPI_Scatterv, а не MPI_Scatter, потому что количество элементов у каждого процесса может быть разным. Функция содержит параметры с буфером передачи, содержащим вектор с элементами, количеством элементов для каждого процесса local_N, вектором со смещенями, MPI_INT типа, процессы приема получают local_array, или буфер приема, который содержит элементы только для конкретного процесса, количество элементов, так же MPI_INT типа, корневой передающий процесс, или root процесс, 0, и коммуникатор MPI_COMM_WORLD. Процессы приема имеют параметры только приема, параметры передачи имеют nullptr и MPI_DATATYPE_NULL. Каждый процесс в цикле for считает локальную сумму в переменную local_sum. С помощью коллективной операции MPI_Reduce происходит сложение всех локальных сумм со всех процессов и на нулевом процессе обновляется general_sum. Функция имеет аргументы с указателем на буфер отправляемого сообщения, а именно локальной суммы, и буфер для результата, общей суммы, количество отправляемых элементов, MPI_INT типа, поскольку необходима сумма - операция MPI_SUM, ранг процесса для приема результата, или нулевой процесс, и коммуникатор MPI_COMM_WORLD. 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_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &ProcNum);
    MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);

    int N = 100001;
    int local_sum = 0;
    int general_sum = 0;

    vector<int> local_N(ProcNum);
    vector<int> displs(ProcNum);
    int quotient = N / ProcNum;
    int remainder = N % ProcNum;

    for (int i = 0; i < ProcNum; i++) {
        local_N[i] = quotient + (i < remainder ? 1 : 0);
        displs[i] = (i == 0) ? 0 : displs[i - 1] + local_N[i - 1];
    }

    vector<int> local_array(local_N[ProcRank]);

    double start, end;

    if (ProcRank == 0) {
        vector<int> array(N, 1);
        start = MPI_Wtime();
        MPI_Scatterv(array.data(),   local_N.data(),   displs.data(),   MPI_INT,
local_array.data(), local_N[ProcRank], MPI_INT, 0, MPI_COMM_WORLD);
    } else {
        MPI_Scatterv(nullptr,      nullptr,      nullptr,      MPI_DATATYPE_NULL,
local_array.data(), local_N[ProcRank], MPI_INT, 0, MPI_COMM_WORLD);
    }

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

    MPI_Reduce(&local_sum,    &general_sum,    1,    MPI_INT,    MPI_SUM,    0,
MPI_COMM_WORLD);

    if (ProcRank == 0) {
        end = MPI_Wtime();
        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 сложение элементов вектора.

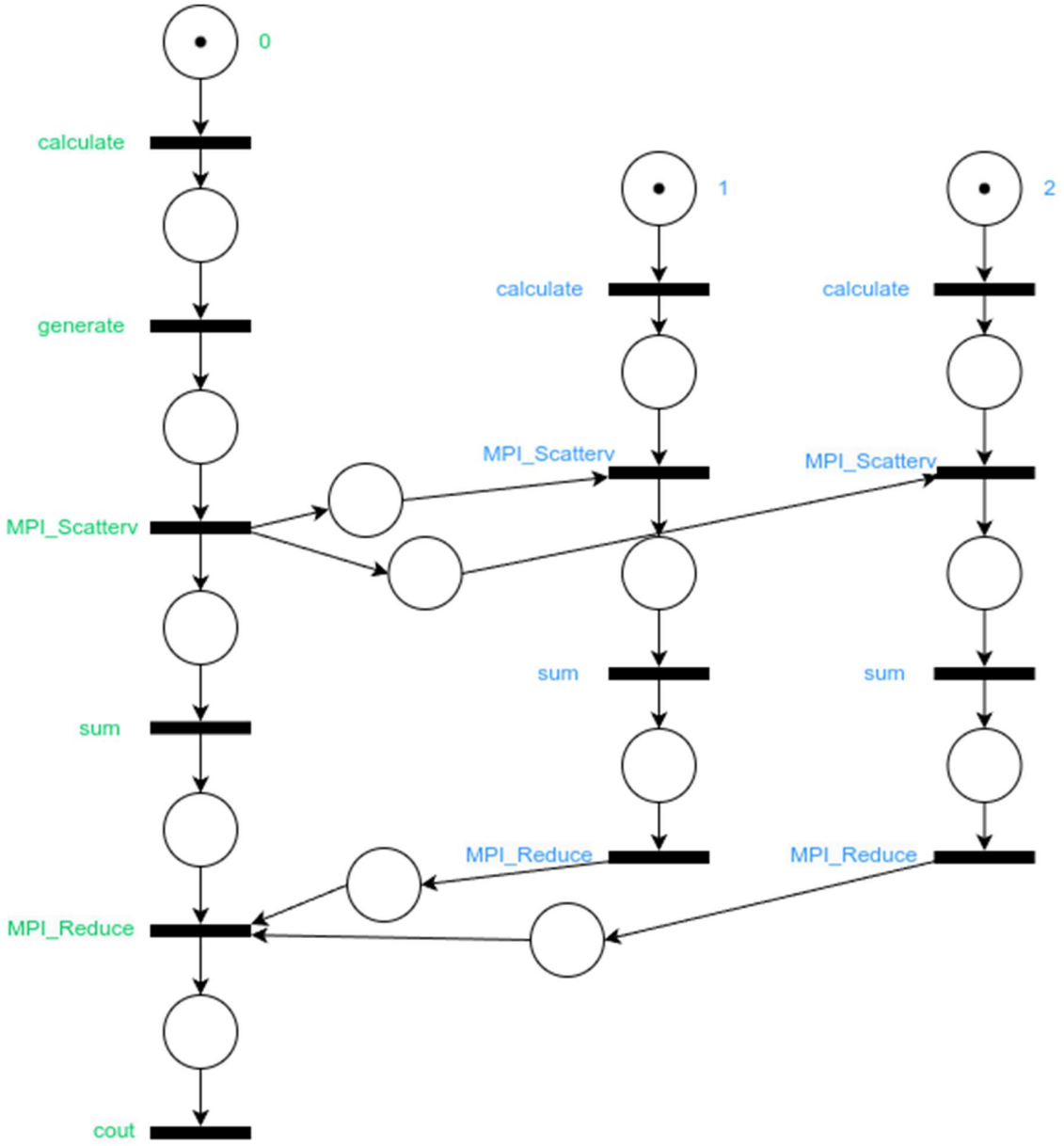

Рисунок 1 – Сеть Петри для параллельной программы сложения элементов вектора для трех процессов

4. Программа была запущена на разных числах процессов: 1, 2, 4, 6, 8, 10, 12, 14, 16 и разным числом элементов в векторе: 100000 и 100. Время

выполнения программы в зависимости от количества процессов представлено со 100000 элементами в векторе в листинге 2, а с 100 – в листинге 3.

Листинг 2.

```
cut@cute:~/Документы/cpluscplus/4$ mpiexec.openmpi -n 1 --oversubscribe ./main
Общая сумма: 100000
Time 0.000623259
cut@cute:~/Документы/cpluscplus/4$ mpiexec.openmpi -n 2 --oversubscribe ./main
Общая сумма: 100000
Time 0.00036546
cut@cute:~/Документы/cpluscplus/4$ mpiexec.openmpi -n 4 --oversubscribe ./main
Общая сумма: 100000
Time 0.000395386
cut@cute:~/Документы/cpluscplus/4$ mpiexec.openmpi -n 6 --oversubscribe ./main
Общая сумма: 100000
Time 0.000386862
cut@cute:~/Документы/cpluscplus/4$ mpiexec.openmpi -n 8 --oversubscribe ./main
Общая сумма: 100000
Time 0.000413558
cut@cute:~/Документы/cpluscplus/4$ mpiexec.openmpi -n 10 --oversubscribe ./main
Общая сумма: 100000
Time 0.000448107
cut@cute:~/Документы/cpluscplus/4$ mpiexec.openmpi -n 12 --oversubscribe ./main
Общая сумма: 100000
Time 0.000600551
cut@cute:~/Документы/cpluscplus/4$ mpiexec.openmpi -n 14 --oversubscribe ./main
Общая сумма: 100000
Time 0.00159347
cut@cute:~/Документы/cpluscplus/4$ mpiexec.openmpi -n 16 --oversubscribe ./main
Общая сумма: 100000
Time 0.002121312
```

Листинг 3.

```
cut@cute:~/Документы/cpluscplus/4$ mpiexec.openmpi -n 1 --oversubscribe ./main
Общая сумма: 100
Time 8.302e-06
cut@cute:~/Документы/cpluscplus/4$ mpiexec.openmpi -n 2 --oversubscribe ./main
Общая сумма: 100
Time 6.6963e-05
cut@cute:~/Документы/cpluscplus/4$ mpiexec.openmpi -n 4 --oversubscribe ./main
Общая сумма: 100
Time 0.000115311
cut@cute:~/Документы/cpluscplus/4$ mpiexec.openmpi -n 6 --oversubscribe ./main
Общая сумма: 100
Time 0.000227885
cut@cute:~/Документы/cpluscplus/4$ mpiexec.openmpi -n 8 --oversubscribe ./main
Общая сумма: 100
Time 0.000256803
cut@cute:~/Документы/cpluscplus/4$ mpiexec.openmpi -n 10 --oversubscribe ./main
Общая сумма: 100
Time 0.000337270
cut@cute:~/Документы/cpluscplus/4$ mpiexec.openmpi -n 12 --oversubscribe ./main
Общая сумма: 100
Time 0.000432860
cut@cute:~/Документы/cpluscplus/4$ mpiexec.openmpi -n 14 --oversubscribe ./main
```

```
Общая сумма: 100
Time 0.000653845
cut@cute:~/Документы/cpluscplus/4$ mpiexec.openmpi -n 16 --oversubscribe ./main
Общая сумма: 100
Time 0.001593956
```

Программа была также запущена с разным количеством элементов в векторе: 100000, 200000, 300000, 400000, 500000, на четырех процессах. Время выполнения программы в зависимости от объема данных представлено в листинге 4.

Листинг 4.

```
cut@cute:~/Документы/cpluscplus/4$ mpicxx.openmpi main.cpp -o ./main && mpiexec.openmpi -n 4 --oversubscribe ./main
Общая сумма: 100000
Time 0.000434463
cut@cute:~/Документы/cpluscplus/4$ mpicxx.openmpi main.cpp -o ./main && mpiexec.openmpi -n 4 --oversubscribe ./main
Общая сумма: 200000
Time 0.000702875
cut@cute:~/Документы/cpluscplus/4$ mpicxx.openmpi main.cpp -o ./main && mpiexec.openmpi -n 4 --oversubscribe ./main
Общая сумма: 300000
Time 0.0011314
cut@cute:~/Документы/cpluscplus/4$ mpicxx.openmpi main.cpp -o ./main && mpiexec.openmpi -n 4 --oversubscribe ./main
Общая сумма: 400000
Time 0.00121495
cut@cute:~/Документы/cpluscplus/4$ mpicxx.openmpi main.cpp -o ./main && mpiexec.openmpi -n 4 --oversubscribe ./main
Общая сумма: 500000
Time 0.00166234
```

5. На основе полученных данных при выполнении программы на разных количествах процессов построены графики зависимости времени выполнения программы от числа запущенных процессов со 100000 и 100 элементами, которые представлены на рис. 2-3 соответственно. Также построен график зависимости времени выполнения программы от количества данных на основе выполнения работы программы с разным количеством элементов в векторе и представлен на рис. 4.

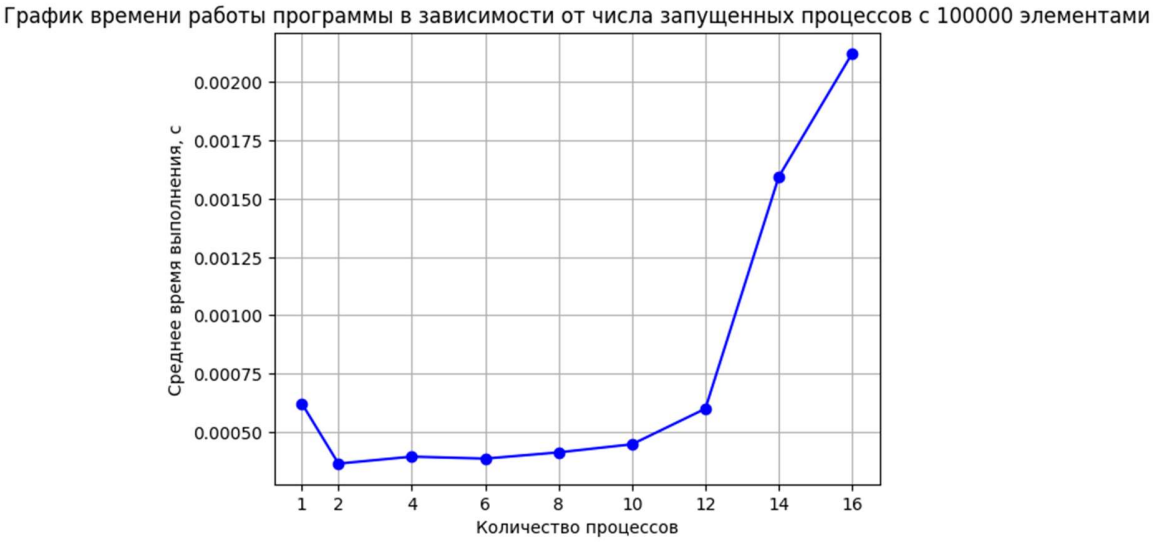

Рисунок 2 – График времени работы программы в зависимости от числа запущенных процессов со 100000 элементами

Рисунок 3 – График времени работы программы в зависимости от числа запущенных процессов с 100 элементами

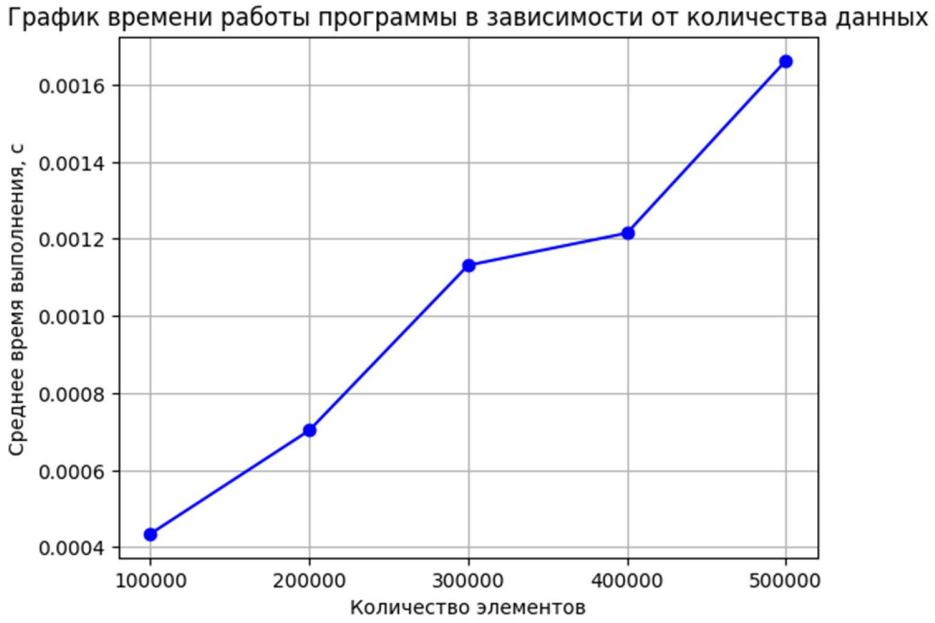

Рисунок 4 – График времени работы программы в зависимости от количества данных

Исходя из графиков, представленных на рис. 2-3, время работы программы увеличивается с ростом количества запущенных процессов при малом числе элементов, и при большом числе элементов наблюдается сокращение времени работы программы по сравнению с работой на одном процессе до некоторого момента, а затем происходит рост времени работы, на что также влияет количество ядер и потоков в системе. В программе в рассмотрении скорости работы программы с коллективными операциями есть две линии: одна связана с ростом количества процессов, а вместе с этим и нагрузкой на распределение элементов, а вторая - с количеством элементов, что несколько процессов могут отработать быстрее в локальных суммах, а затем и в коллективном сложении, даже с учетом распределения данных между ними. С увеличением количества элементов, исходя из графика, представленного на рис. 4, возрастает время работы программы, что связано с большим числом элементов, которые надо передать и над которыми необходимо провести операции. Время выполнения программы так же связано с распределением

элементов между процессами, как в случае с остатком элементов некоторые процессы загружаются больше остальных. Время выполнения программы может быть разным при одинаковых условиях, что может быть вызвано как разной скоростью деления и раздачи данных, выполнения процессов, сбора данных, 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 1 процессом на время выполнения программы с 2, 4, 6, 8, 10, 12, 14, 16 процессами соотвественно. Графики ускорения/замедления представлены на рис. 5-6 с 100000 и 100 элементами соответственно.

Рисунок 5 – График ускорения/замедления работы программы с разным количеством запущенных процессов с 100000 элементами

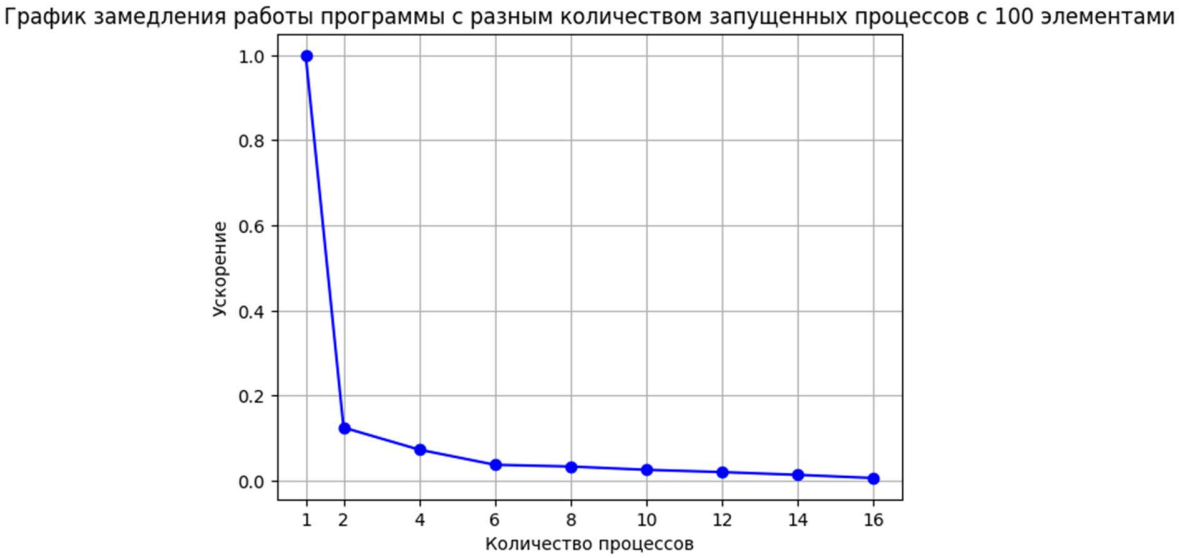

Рисунок 6 – График замедления работы программы с разным количеством запущенных процессов с 100 элементами

Для построения графика ускорения/замедления работы программы с разным количеством элементов необходимо разделить время выполнения программы с 100000 элементами в векторе на время выполнения программы с 200000, 300000, 400000, 500000 элементами соотвественно. График замедления представлен на рис. 7.

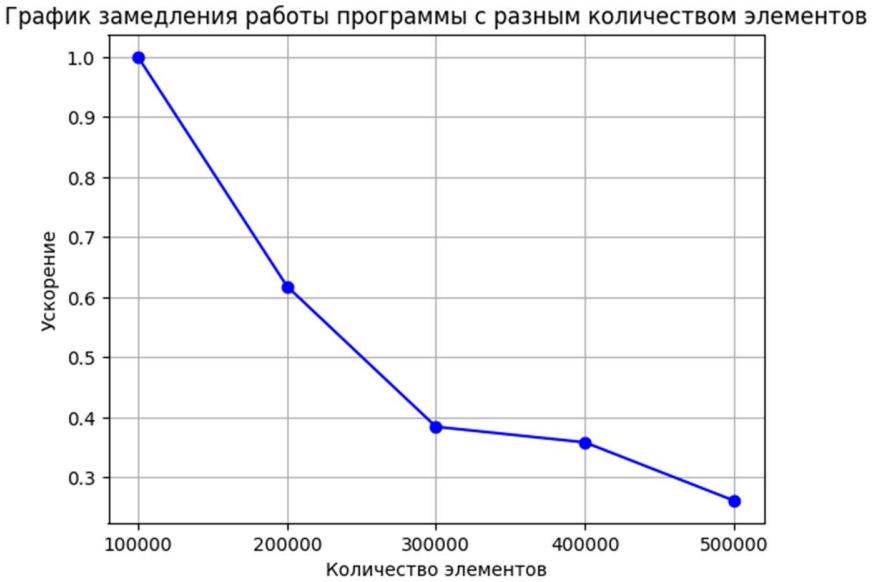

Рисунок 7 – График замедления работы программы с разным количеством элементов

Вывод.

Таким образом, было освоено написание параллельных программ MPI с использованием коллективных операций библиотеки MPI.

Были изучены зависимости времени работы программы от числа запущенных процессов и количества элементов в векторе. С увеличением числа процессов время работы программы увеличивается на малом числе элементов, поскольку распределение и сбор данных при увеличении числа процессов повышает нагрузку, однако на большом числе элементов наблюдается ускорение в некотором промежутке количества процессов по сравнению с одним процессом, что связано с преобладанием скорости в случае коллективных операций и локальных сумм на нескольких процессах благодаря MPI. Также время выполнения программы возрастает при увеличении количества элементов в векторе, что связано с увеличением объема данных для распределенной передачи и вычислительных операций. 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 ускорения/замедления работы программы. Для программы была построена сеть Петри.