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: Использование функций обмена данными «точка-точка» в библиотеке MPI

Студент гр. 8303 _____ Почаев Н.А.

Преподаватель _____ Татаринов Ю.С.

Санкт-Петербург

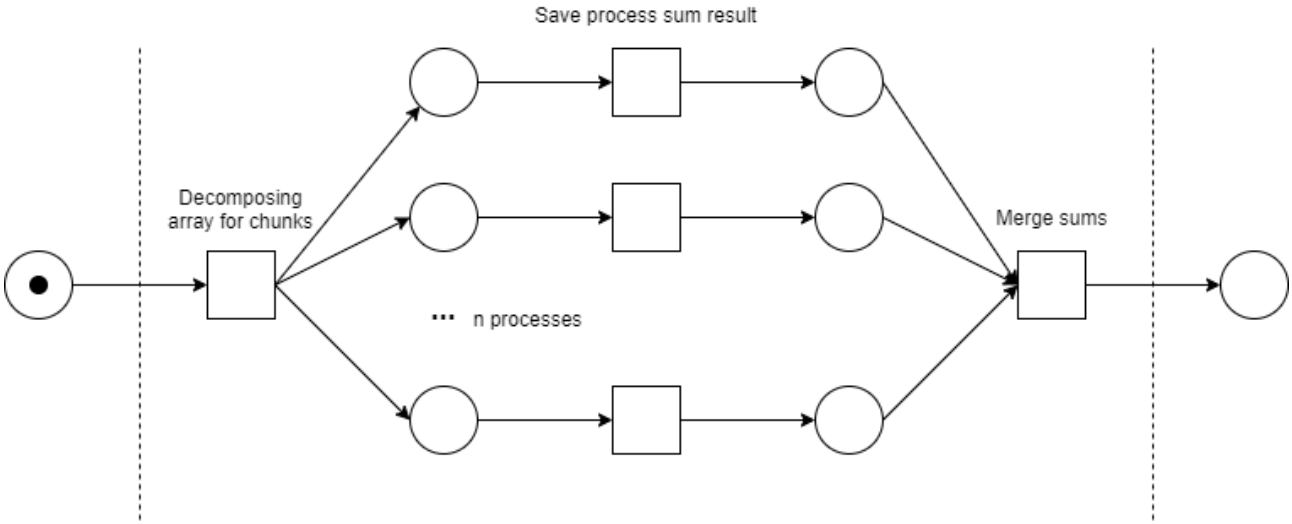
2020

Задание.

Вариант 6 - Суммирование элементов массива. Процесс 0 генерирует массив и раздает его другим процессам для вычисления локальных сумм, после чего вычисляет общую сумму.

Сети Петри.

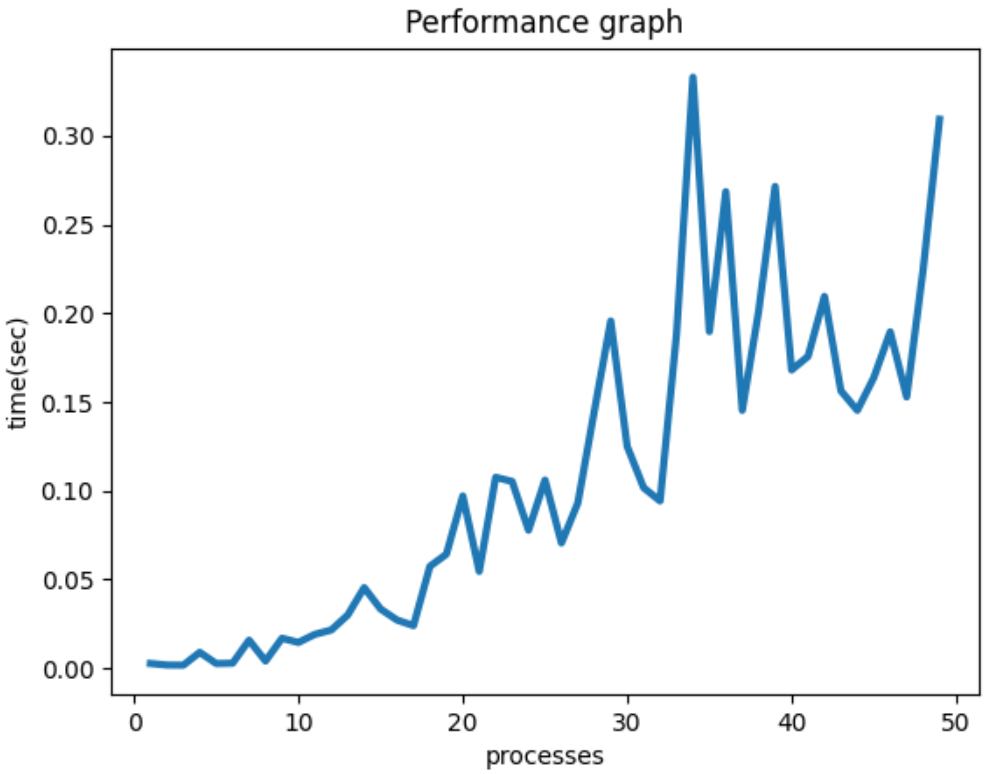
Рассматривается единичный процесс – master, который рассылает единовременно $n-1$ число процессов – slaves, задание по вычислению определённого фрагмента массива. Следовательно, сеть Петри будет выглядеть следующим образом:



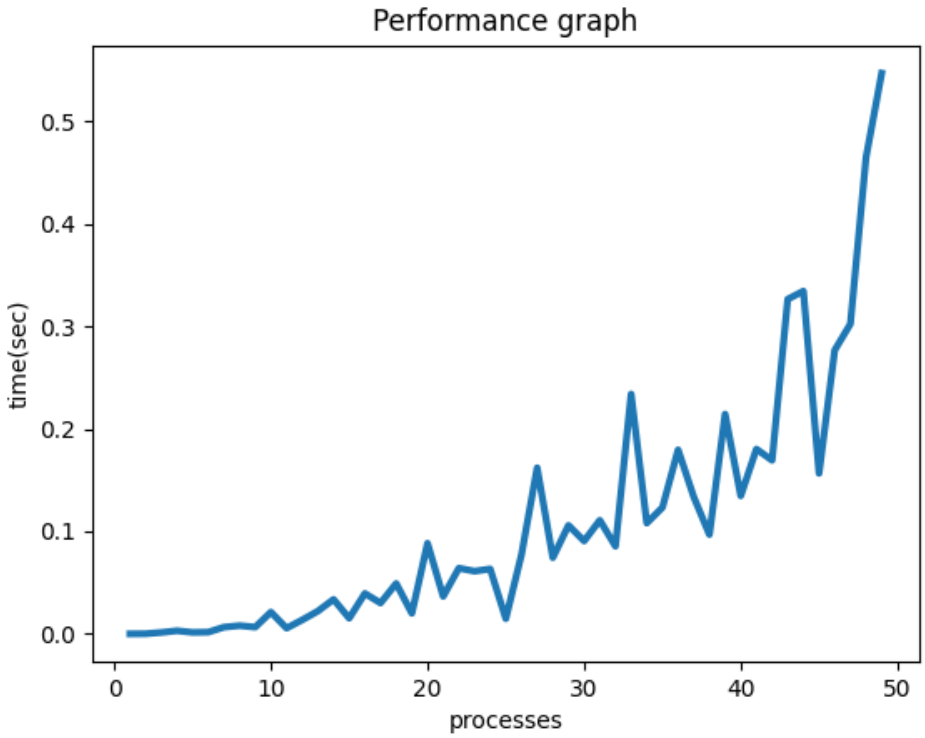
Описание решения.

Процесс №0 создаёт массив чисел указанной длинны и при помощи функции широковещательной рассылки данный MPI_Cast рассылает другим доступным процессам требуемые им для расчёта части массива. Далее master процесс дожидается результатов подсчётов других процессов и с помощью функции MPI_Reduce вычисляет результирующую сумму. Подсчёт для указанного массива и числа процессов, с которым запущена программа, выполняется несколько раз для корректного подсчёта среднего времени выполнения.

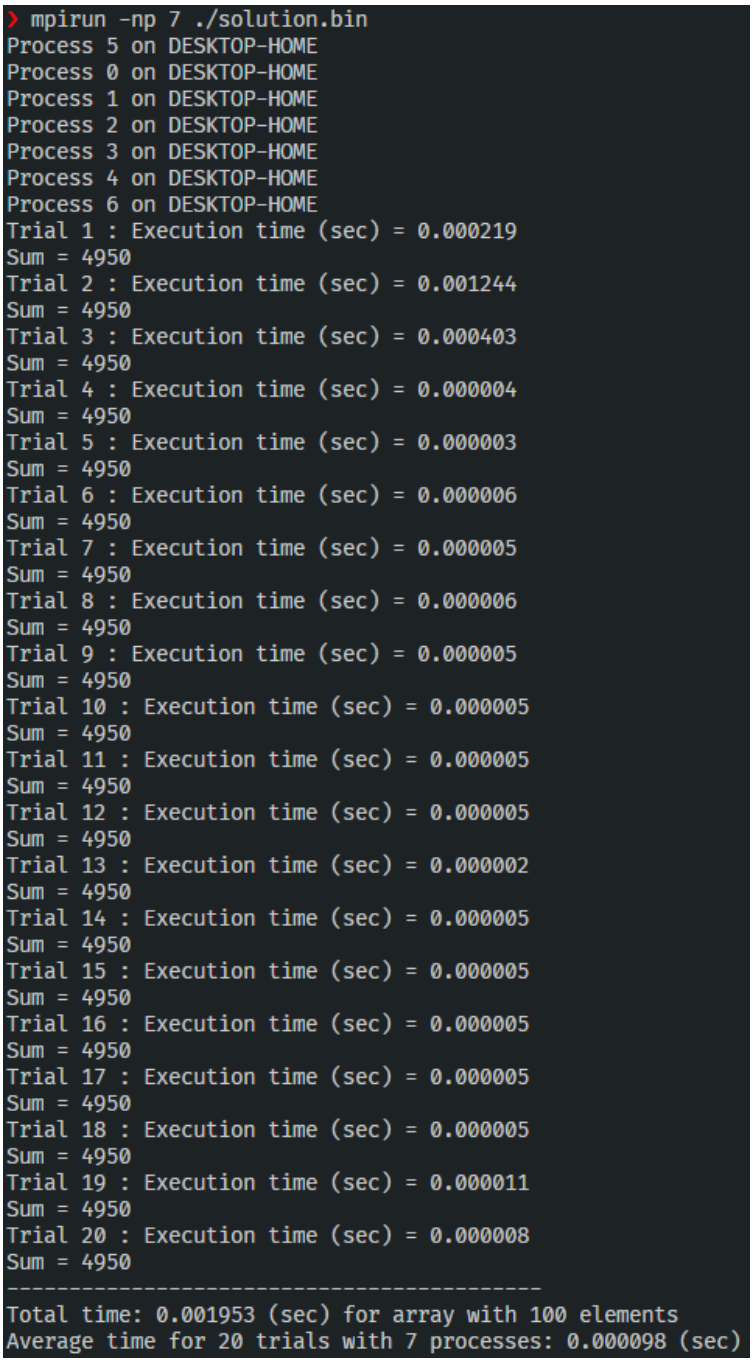
График зависимости времени от числа доступных процессов для массива длиной 10000 элементов и 100 попыток представлен ниже:



Далее представлен график для массива из 100 элементов с 20 попытками на измерение:



Результаты запуска программы представлены на скриншотах ниже:



В данном случае программа запускалась с 7 доступными процессами в ручном режиме (без bash скрипта).

Выводы.

В ходе выполнения лабораторной работы лабораторной работы была изучена работа с процессами и приёмы передачи между ними с использованием API MPI. Экспериментально были получены графики зависимости времени

выполнения параллельных вычислений от числа доступных процессов. Из них возможно сделать вывод, что в данном случае наблюдается тенденция роста времени работы программы от количества процессов, выполняющих её. Связано это с накладными ресурсами на распараллеливание и на ожидание завершения каждого процесса. Учитывая, что операция сложения выполняется чрезвычайно быстро на современных процессорах, параллельное выполнение данной задачи на числах, пусть и больших, но обозримых, является нерациональным.

ПРИЛОЖЕНИЕ А.
ИСХОДНЫЙ КОД ПРОГРАММЫ.

```
#include "mpi.h"
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#define TRIALS 100
#define ARRAY_SIZE 10000

// #define FULL_PRINT
#define TEST

int main(int argc, char *argv[]) {
    int myid, numprocs;
    double startwtime, endwtime;
    int namelen;
    unsigned long long* numbers = new unsigned long
long[ARRAY_SIZE];
    unsigned long long i, j, sum, part_sum;
    unsigned long long startIndex, endIndex;
    int s, s0;
    double totalTime;

    char processor_name[MPI_MAX_PROCESSOR_NAME];

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &myid);
    MPI_Get_processor_name(processor_name, &namelen);

    #ifdef FULL_PRINT
    fprintf(stderr, "Process %d on %s\n", myid, processor_name);
    #endif

    fflush(stderr);

    for (i = 0; i < ARRAY_SIZE; i++) {
        numbers[i] = i;
    }

    if (myid == 0) {
        s = (int) floor(ARRAY_SIZE / numprocs);
        s0 = s + ARRAY_SIZE % numprocs;

        //printf("s=%d , s0= %d\n", s, s0);
    }

    MPI_Bcast(&s, 1, MPI_INT, 0, MPI_COMM_WORLD);
```

```
    MPI_Bcast(&s0, 1, MPI_INT, 0, MPI_COMM_WORLD);

    startIndex = s0 + (myid - 1) * s;
    endIndex = startIndex + s;
    totalTime = 0;

    for (j = 1; j <= TRIALS; j++) {
        if (myid == 0) {
            startwtime = MPI_Wtime();
        }

        sum = 0;
        part_sum = 0;

        if (myid == 0) { // master
            // compute sum of master's numbers
            for (i = 0; i < s0; i++) {
                part_sum += numbers[i];
            }
        } else {
            for (i = startIndex; i < endIndex; i++) {
                part_sum += numbers[i];
            }
        }

        MPI_Reduce(&part_sum,  &sum,  1,  MPI_INT,  MPI_SUM,  0,
MPI_COMM_WORLD);

        if (myid == 0) {
            double runTime;
            endwtime = MPI_Wtime();
            runTime = endwtime - startwtime;

            #ifdef FULL_PRINT
            printf("Trial %llu : Execution time (sec) = %f\n", j,
runTime);
            printf("Sum = %llu \n", sum);
            #endif

            totalTime += runTime;
        }
    } // end for

    if (myid == 0) {
        #ifdef FULL_PRINT
        printf("-----------------------------------------\n");
        printf("Total time: %f (sec) for array with %d elements\n",
totalTime, ARRAY_SIZE);
        printf("Average time for %d trials with %d processes: %f
(sec)\n", TRIALS, numprocs, totalTime / TRIALS);
        #endif
        #ifdef TEST
```

```
        printf("%d %f\n", numprocs, totalTime);
        #endif
    }

    MPI_Finalize();
}
```