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

ОТЧЕТ

по лабораторной работе №3
по дисциплине «Параллельные алгоритмы».
Тема: Коллективные операции.

Студентка гр. 0382 _____ Кривенцова Л.С.

Преподаватель _____ Татаринов Ю.С.

Санкт-Петербург
2022

Задание.

Вариант 3.

В главном процессе дан набор из 3K чисел, где K — количество процессов. Используя функцию MPI_Scatter, переслать по 3 числа в каждый процесс (включая главный) и вывести в каждом процессе полученные числа.

Листинг программы см. в Приложении А.

Выполнение работы.

В программе осуществляется пересылка чисел массива во все участвующие процессы (включая отправитель). Если процесс нулевой, то в выделенную ранее память для массива размером [количество процессов * 3] записываются числа, генерируемые с помощью rand(). Адрес этого массив затем передаётся в функцию MPI_Scatter в качестве первого аргумента. Перед вызовом функции инициализируется ещё один массив, адрес которого так же будет передан в функцию в качестве начала буфера приема. Также в функцию передаётся количество элементов (в нашем случае, чисел) в отправляемом/получаемом блоке и информацию и о типе пересылаемых данных.

MPI_Scatter(array, 3, MPI_INT, numbers, 3, MPI_INT, 0, MPI_COMM_WORLD);

Номер процесса и полученные числа выводятся в консоль.

Результаты работы программы на 1,2 …. N процессах.

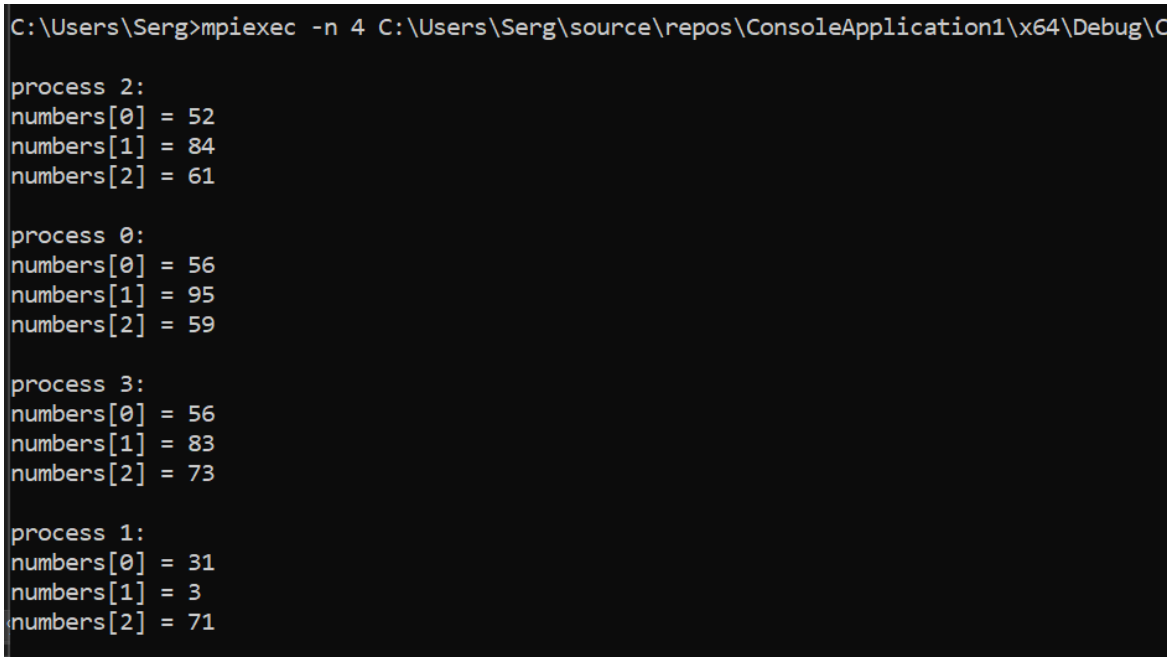

Рис. 1 - Результаты работы программы на 4 процессах.

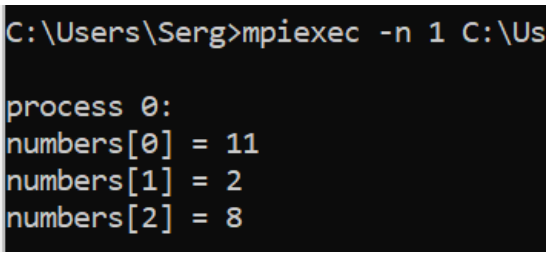

Рис. 2 - Результаты работы программы на 1 процессе.

Пересылка осуществляется только «самому себе» за неимением другиъ процессов, и длина массива равна трём.

График зависимости времени выполнения программы от числа процессов для разных длин пересылаемых сообщений.

Эксперимент проводится без учёта разной длины массива, т.к. она вычисляется напрямую от числа процессов.

Для получения экспериментальных данных модифицируем программу так, чтобы выводились также нужные для построения графика данные: время работы программы на каждом процессоре.

Для выявления зависимости получим значения времени каждого запуска программы с различным количеством процессов:

Номер процесса	Время выполнения программы, с
0	0,000062
Среднее	0,000062
0	0,000141
1	0,00021
Среднее	0,0001755
0	0,000301
1	0,000363
2	0,000418
3	0,000479
4	0,000287
Среднее	0,0003696
0	0,000524
1	0,000701
2	0,000711
3	0,00086
4	0,000682
5	0,000839
6	0,00084
7	0,000952
8	0,000545
9	0,000714
Среднее	0,0007368

Рис.3 – Экспериментальные данные запусков программы на 0,1,2,5,10 процессах.

Число процессов	Время выполнения
0	0
1	0,000062
2	0,0001755
5	0,0003696
10	0,0007368

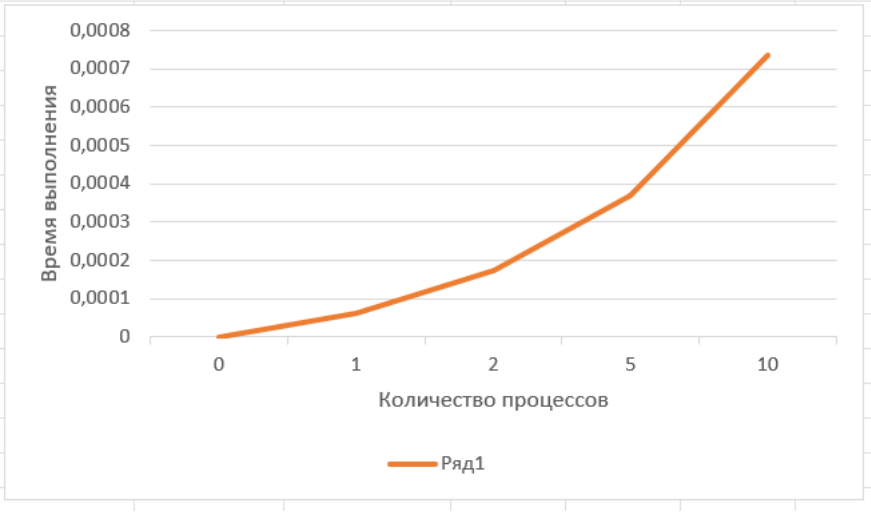

Рис.4 – График зависимости времени выполнения программы от числа процессов

(и данные для построения).

Следовательно, выполняемое время напрямую зависит от количества зависимых процессов: чем больше процессов, тем дольше идёт работа с пересылом данных. Это происходит, потому что чем больше процессов, тем больше работы для программы: повышается количество выполняемых действий (отправок и приёмов сообщения).

Сети Петри.

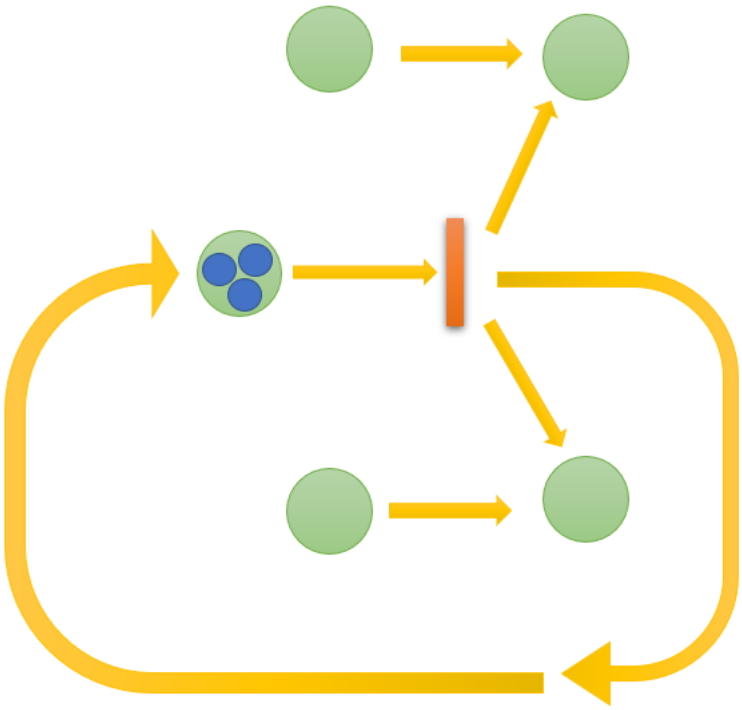

Рис.5 – Сети Петри.

Выводы.

Написана программа обмена сообщениями, осуществляющая пересылку массива по частям от процесса 0 во все процессы (включая себя).

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#include <stdio.h>
#include <mpi.h>
#include <cstdlib>
#include <ctime>
#define MAX_LENGTH 1000

int main(int argc, char* argv[])
{
  srand(time(0));
  double time_start, time;
  int size, rank, count = 0, index = 0, index_res = 0;
  double double_x;
  MPI_Status status;
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);

  int  rbuf[3];
  int  root = rank;
  int* array = new int[size * 3];
  time_start = MPI_Wtime();
  if (rank == 0) {
    for (int i = 0; i < size * 3; i++) {
      array[i] = rand() % 100;
    }
  }

  int* numbers = (int*)malloc(sizeof(int) * 3);

  MPI_Scatter(array, 3, MPI_INT, numbers,
    3, MPI_INT, 0, MPI_COMM_WORLD);
  printf("\nprocess %d:\n", rank);
  for (int c = 0; c < 3; c++)
    printf("numbers[%d] = %d\n", c, numbers[c]);
  MPI_Finalize();
  return 0;
}
```