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

ОТЧЕТ

по лабораторной работе №4
по дисциплине «Параллельные алгоритмы».
Тема: Группы процессов и коммуникаторы.

Студентка гр. 0382 _____ Кривенцова Л.С.

Преподаватель _____ Татаринов Ю.С.

Санкт-Петербург
2022

Задание.

Вариант 10.

В каждом процессе дано целое число N, которое может принимать два значения: 1 и 2 (имеется хотя бы один процесс с каждым из возможных значений). Кроме того, в каждом процессе дано целое число A. Используя функцию *MPI_Comm_split* и одну коллективную операцию пересылки данных, переслать числа A, данные в процессах с N = 1, во все процессы с N = 1, а числа A, данные в процессах с N = 2, во все процессы с N = 2. Во всех процессах вывести полученные числа в порядке возрастания рангов переславших их процессов (включая число, полученное из этого же процесса).

Листинг программы см. в Приложении А.

Выполнение работы.

В программе создаётся новый коммуникатор (*newcomm*), с помощью которого будет осуществляться пересыл данных внутри групп процессов. Создаётся (и инициализируется нулями) динамический массив, в который будут записываться собранные числа (массив всех чисел A).

Целое число A вычисляется в каждом процессе как [его номер + 1]. Это сделано для наглядности пересыла, а число A может являться совершенно любым числом (например, генерировать случайное число в диапазоне целых чисел). Далее задаётся число N: в нулевом процессе оно равно 1, в первом – 2, так как это позволяет соблюсти условие, что имеется хотя бы один процесс с каждым из возможных значений. Во всех остальных процессах это число выбирается случайным образом – от 1 до 2.

Вызывается функция *MPI_Comm_split*. Она логически разбивает процессы по группам (в качестве критерия («цвета») выступает значение N) и имеет структуру:

MPI_Comm_split(MPI_COMM_WORLD, n, rank, &newcomm);

Где MPI_COMM_WORLD – родительский коммуникатор, n – признак группы, переменная *rank* определяет упорядочивание, а *&newcomm* – адрес нового коммуникатора.

Затем происходит вызов функции *MPI_Allgather*, которая позволяет переслать числа A, данные в процессах определенной подгруппы, во все процессы этой подгруппы. Функция имеет структуру:

*MPI_Allgather(&a, 1, MPI_INT, *&array, 1, MPI_INT, newcomm);*

Где *&a* - адрес начала размещения посылаемых данных (значение A процесса), 1 и 1 – числа посылаемых и получаемых элементов, *MPI_INT* – тип этих элементов, *array* - начало буфера приема, *newcomm* – коммуникатор.

Номер процесса и полученные им числа выводятся в консоль.

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

```
C:\Users\Serg>mpiexec -n 4 C:\Users\Serg\source\repos\ConsoleApp
rank = 3, N = 2, A = 4
Received counts:
a[0] = 2
a[1] = 3
a[2] = 4
rank = 1, N = 2, A = 2
Received counts:
a[0] = 2
a[1] = 3
a[2] = 4
rank = 0, N = 1, A = 1
Received counts:
a[0] = 1
rank = 2, N = 2, A = 3
Received counts:
a[0] = 2
a[1] = 3
a[2] = 4
```

Рис. 1 - Результаты работы программы на 4 процессах.

```
C:\Users\Serg>mpiexec -n 2 C:\Users\Serg\source\repos\Co
rank = 1, N = 2, A = 2
Received counts:
a[0] = 2
rank = 0, N = 1, A = 1
Received counts:
a[0] = 1
```

Рис. 2 - Результаты работы программы на 2 процессах.

Пересылка осуществляется только «самому себе» за неимением других процессов в группе, и длина массива равна одному.

График зависимости времени выполнения программы от числа

процессов.

Так как длина сообщения фиксированная (одно число A), исследуем только зависимость времени 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 этого добавим в программу следующие строчки:

```
time_start = MPI_Wtime();
...
if (rank == 0) printf("Time: %f", MPI_Wtime() - time_start);
```

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

Число процессов	Время выполнения
0	0
2	0,000288
5	0,000755
10	0,001359
20	0,002813

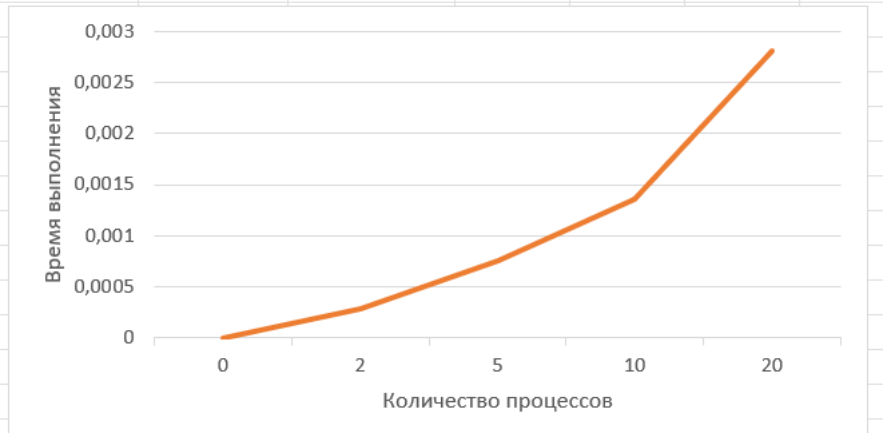

Рис.3 – График зависимости времени выполнения программы от числа процессов (и данные для построения).

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.

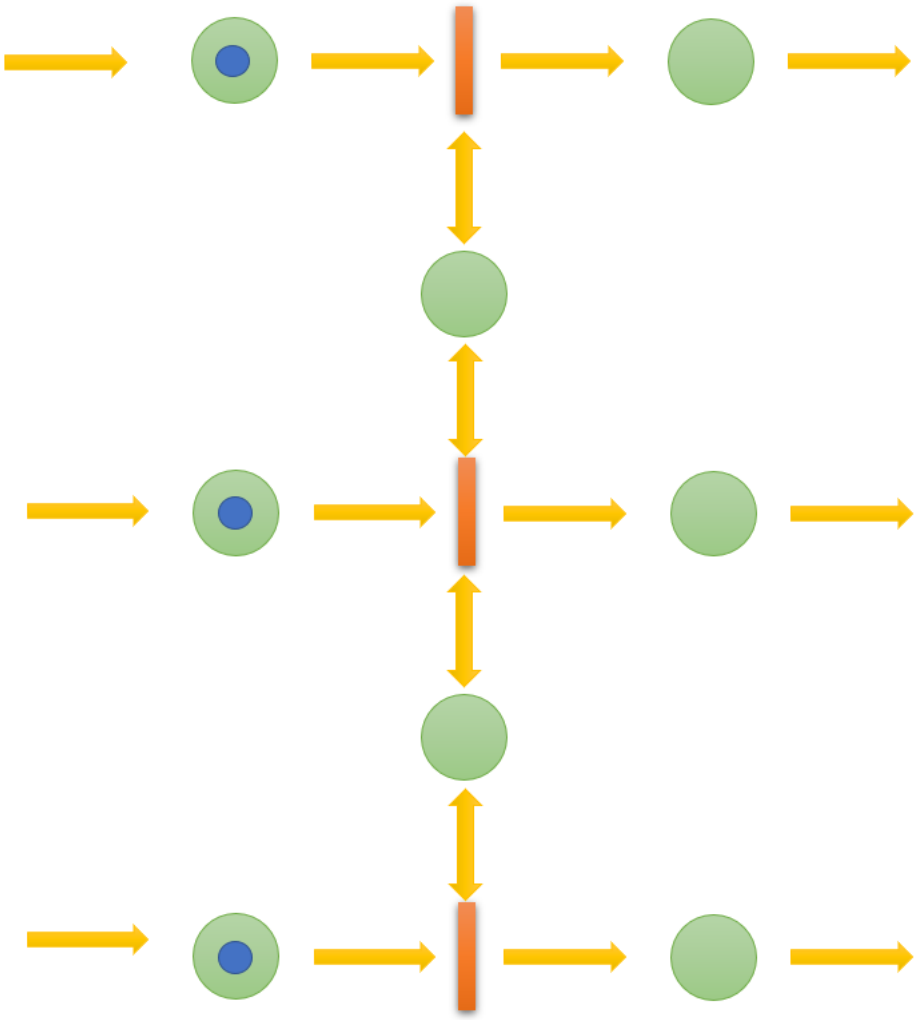

Рис.4 – Сети Петри, на которых отображен процесс пересыла сообщениями (MPI_Allgather) на примере трёх процессов, относящихся к одной группе.

Выводы.

Написана программа, осуществляющая сбор данных по определенным группам процессов с использованием функции расщепления коммуникатора MPI_Comm_split.

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, n, a, newrank, new_a;
  MPI_Status status;
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  int* array = new int[size];
  for (int i = 0; i < size; i++) array[i] = 0;
  MPI_Comm newcomm;
  a = rank + 1;
  time_start = MPI_Wtime();
  if (rank == 0) {
    n = 1;
  }
  else if (rank == 1) n = 2;
  else n = rand() % 2 + 1;

  MPI_Comm_split(MPI_COMM_WORLD, n, rank, &newcomm);
  printf("rank = %d, N = %d, A = %d\n", rank, n, a);
  printf("Received counts:\n");
  int root = 0;
  MPI_Allgather(&a, 1, MPI_INT, *&array, 1, MPI_INT, newcomm);
  for (int i = 0; array[i] > 0; i++) {
    printf("a[%d] = %d\n", i, array[i]);
  }

  MPI_Finalize();
  return 0;
}
```