**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**

Студент гр. 9383 _____ Лапина А.А.

Преподаватель _____ Татаринов Ю.С.

Санкт-Петербург

2021

**Цель работы.**

Написать программу, где необходимо создать новый коммутатор и выполнить операцию сложения.

**Формулировка задания.**

**Вариант 11.**

В каждом процессе дано целое число N, которое может принимать два значения: 0 и 1 (имеется хотя бы один процесс с N = 1). Кроме того, в каждом процессе с N = 1 дано вещественное число A. Используя функцию MPI_Comm_split и одну коллективную операцию редукции, найти сумму всех исходных чисел A и вывести ее во всех процессах с N = 1.

Указание. При вызове функции MPI_Comm_split в процессах, которые не требуется включать в новый коммуникатор, в качестве параметра color следует указывать константу MPI_UNDEFINED.

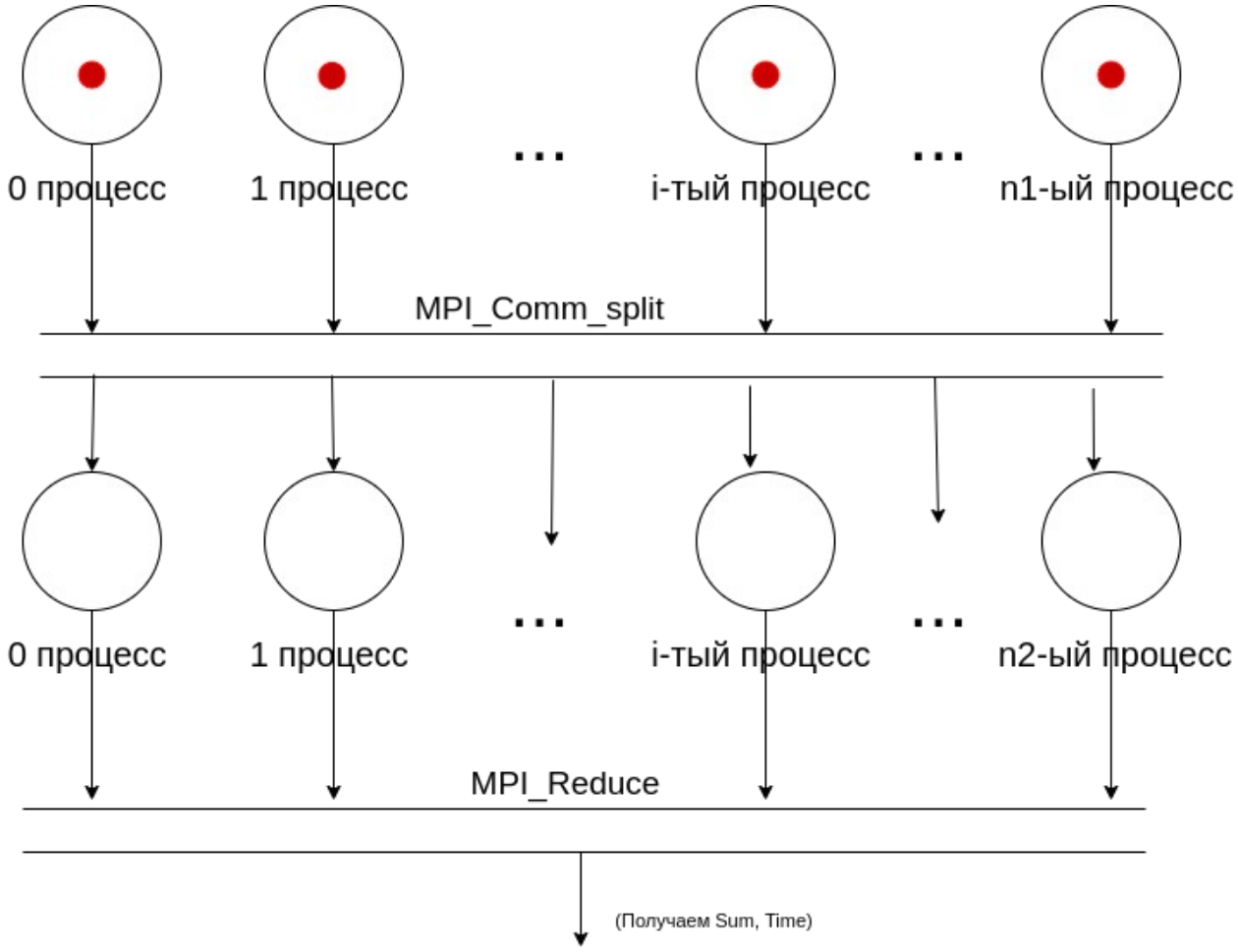
**Краткое описание выбранного алгоритма решения задачи.**

Создаем структуру struct_ для хранения чисел N и A. Задаем массив структур array. Создаем новый коммуникатор Comm с помощью MPI_Comm_split, куда включаем те процессы, где N = 1 (по заданию). Затем переменную NewProcRank, где будем указывать ранг нового коммуникатора. После, используя новый коммуникатор, выполняем коллективную операцию редукции — MPI_Reduce с операцией MPI_Sum — поиска суммы A. В главном процессе нового коммуникатора выводим сумму и время выполнения программы.

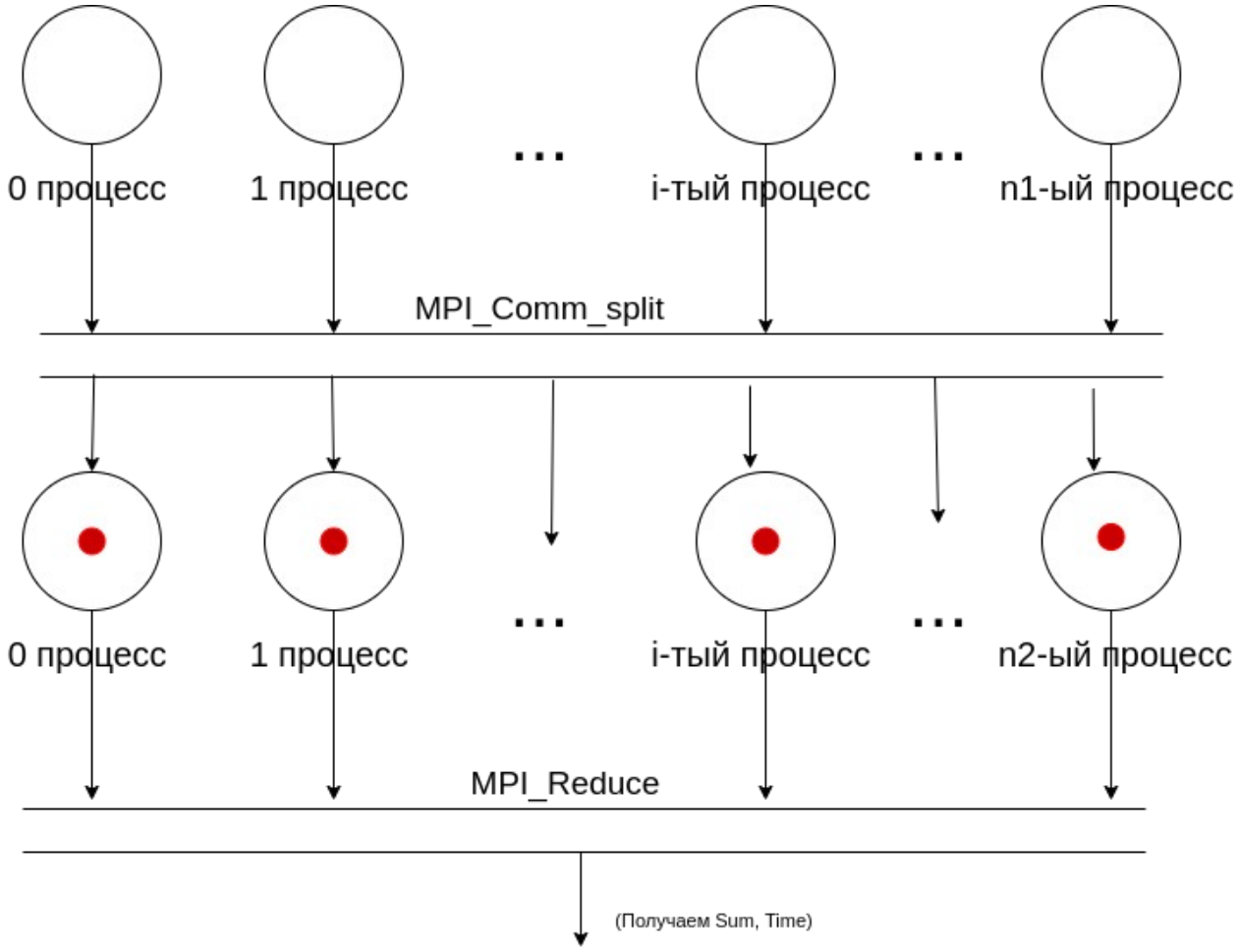
**Формальное описание алгоритма с использованием аппарата Сетей Петри для n процессов:**

n2 — количество всего процессов (в MPI_COMM_WORLD), n1 - количество процессов в новом коммуникаторе, при этом n1>=n2 (зависит от значения N):

Шаг 1 — создаем новый коммуникатор с помощью MPI_Comm_split:



Следующий шаг: После создания нового коммутатора выполняем сложение с помощью MPI_Reduce




***Листинг программы.***

```
#include <stdio.h>
#include "stdlib.h"
#include <mpi.h>
#include "time.h"

struct struct_{
   int N;
   float A;
};

void main( int argc, char *argv[] ) {
   srand(time(NULL));
   float Sum = 0;
   struct struct_ array[4] = {{0}, {1, 0.7}, {1, 0.3}, {0}};
   int ProcRank, ProcNum;
   MPI_Init(&argc, &argv);
   MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);
   MPI_Comm_size(MPI_COMM_WORLD, &ProcNum);
```

```
    int Flag;
    MPI_Initialized(&Flag);
    if (Flag == 0)
        return;

    double Start = MPI_Wtime();
    MPI_Comm Comm;
    int Color;
    if(array[ProcRank].N == 1)
        Color = 0;
    else
        Color = MPI_UNDEFINED;
    MPI_Comm_split(MPI_COMM_WORLD, Color, ProcRank, &Comm);
    if (Comm == MPI_COMM_NULL)
        return;

    int NewProcRank;
    MPI_Comm_rank(Comm, &NewProcRank);

    MPI_Reduce(&array[ProcRank].A, &Sum, 1, MPI_FLOAT, MPI_SUM, 0, Comm);

    if(NewProcRank == 0){
        double Finish = MPI_Wtime();
        double Time = Finish-Start;
        printf("Sum = %f\n", Sum);
        printf("Time: %f\n", Time);
    }
    MPI_Finalize();
}
```

**Результаты работы программы**

1) для 2 процессов и  array[2] = {{0}, {1, 0.555}};;

```
white-lilac@white-lilac:~/Документы/ParAlg/lb4$ mpirun -np 2 --oversubscribe -quiet ./main.x
Sum = 0.555000
Time: 0.000173
```

2) для 4 процессов и array[4] = {{0}, {1, 0.7}, {1, 0.3}, {0}};

```
white-lilac@white-lilac:~/Документы/ParAlg/lb4$ mpirun -np 4 --oversubscribe -quiet ./main.x
Sum = 1.000000
Time: 0.000188
```

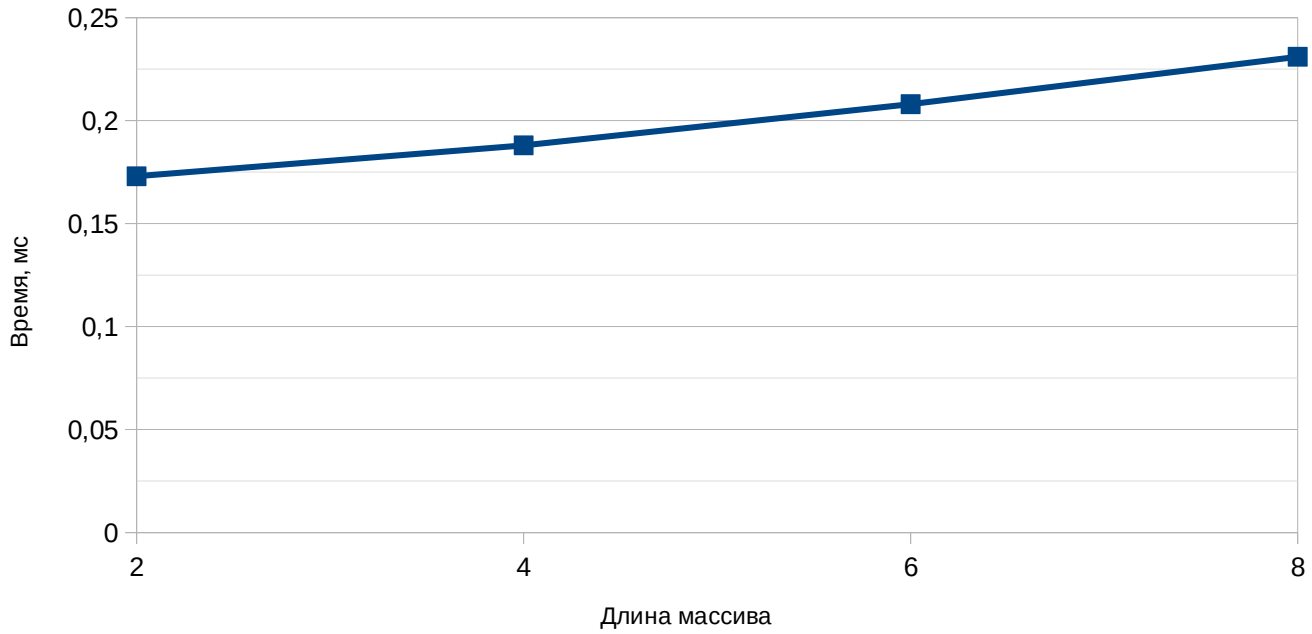
3) для 6 процессов и array[6] = {{0}, {1, 0.111}, {1, 0.3}, {0}, {1, 0.9}, {0}};

```
white-lilac@white-lilac:~/Документы/ParAlg/lb4$ mpirun -np 6 --oversubscribe -quiet ./main.x
Sum = 1.311000
Time: 0.000208
```

4) для 8 процессов и array[8] = {{1, 0.33}, {0}, {0}, {0}, {1, 0.11}, {0}, {1, 0.22}, {0}};

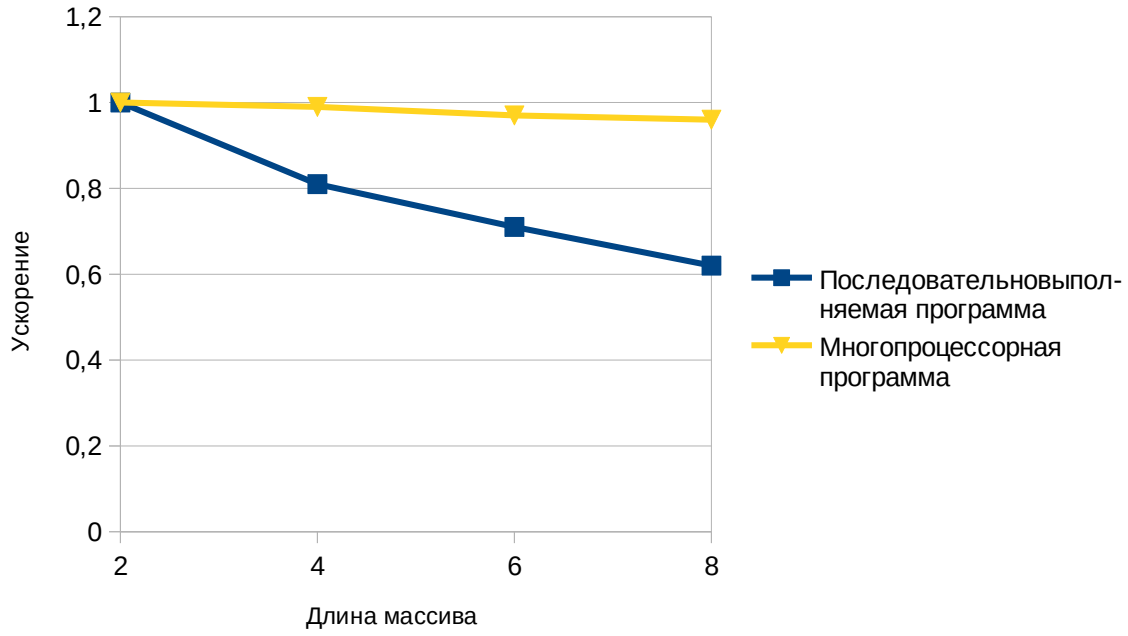
```
white-lilac@white-lilac:~/Документы/ParAlg/lb4$ mpirun -np 8 --oversubscribe -quiet ./main.x
Sum = 0.660000
Time: 0.000231
```

**График зависимости времени выполнения программы от разных длин массива.**




При увеличении количества процессов соответственно возрастает и длина массива array(количество данных для обработки), следовательно время выполнения программы незначительно возрастает. Это происходит потому что мы строим новый коммуникатор и необходимо обработать больше данных, но благодаря распараллеливанию программы время увеличивается не сильно.

**График ускорения (эффективности):**




Для однопроцессорной программы ускорение убывает быстрее, чем с программой использующей несколько процессов. Это происходит потому что при использовании нескольких процессов программа распараллеливается и процессы выполняются одновременно, в отличии от однопроцессорной программы, где все действия, такие как: проверка числа N (является ли оно 1), если да, то включаем A в сумму.

**Вывод.**

Была написана программа с созданием нового коммутатора и была выполнена операция сложения.