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

ОТЧЕТ

по лабораторной работе №5

по дисциплине «Параллельные алгоритмы»

Тема: Виртуальные топологии

Студент гр. 9383 _____ Лапина А.А.

Преподаватель _____ Татаринов Ю.С.

Санкт-Петербург

2021

Цель работы.

Изучить виртуальные топологии и написать программу определения виртуальной топологии.

Формулировка задания.

Вариант 2.

В главном процессе дано целое число N (> 1), не превосходящее количества процессов K. Переслать число N во все процессы, после чего определить декартову топологию для начальной части процессов в виде двумерной решетки — матрицы размера N×K/N (символ «/» обозначает операцию деления нацело, порядок нумерации процессов следует оставить прежним). Для каждого процесса, включенного в декартову топологию, вывести его координаты в этой топологии.

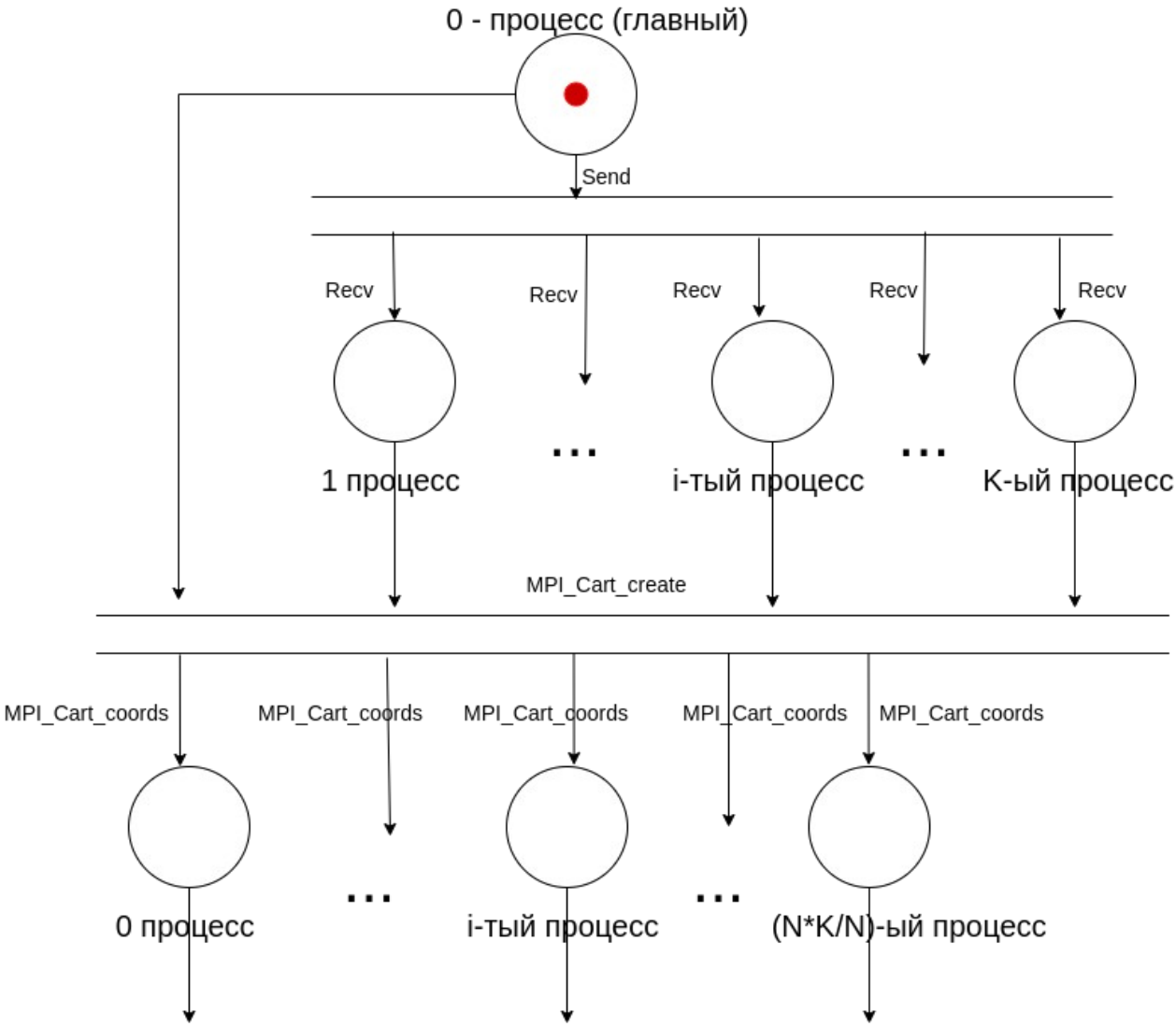
Краткое описание выбранного алгоритма решения задачи.

Число N задается пользователем и оно больше 1, но не превосходит количество процессов K. Главный процесс — нулевой пересылает его всем остальным процессам, после чего определяем декартову топологию для начальной части процессов с помощью MPI_Cart_create. Топология представляет из себя двумерную решетку — матрицу размера N*K/N (деленное нацело). Переменная Dims[] хранит в себе размерность матрицы. Затем с помощью MPI_Cart_coords определяем координаты для процесса (если он включен в декартову топологию), выводим информацию на экран. Подсчитываем время выполнения программы.

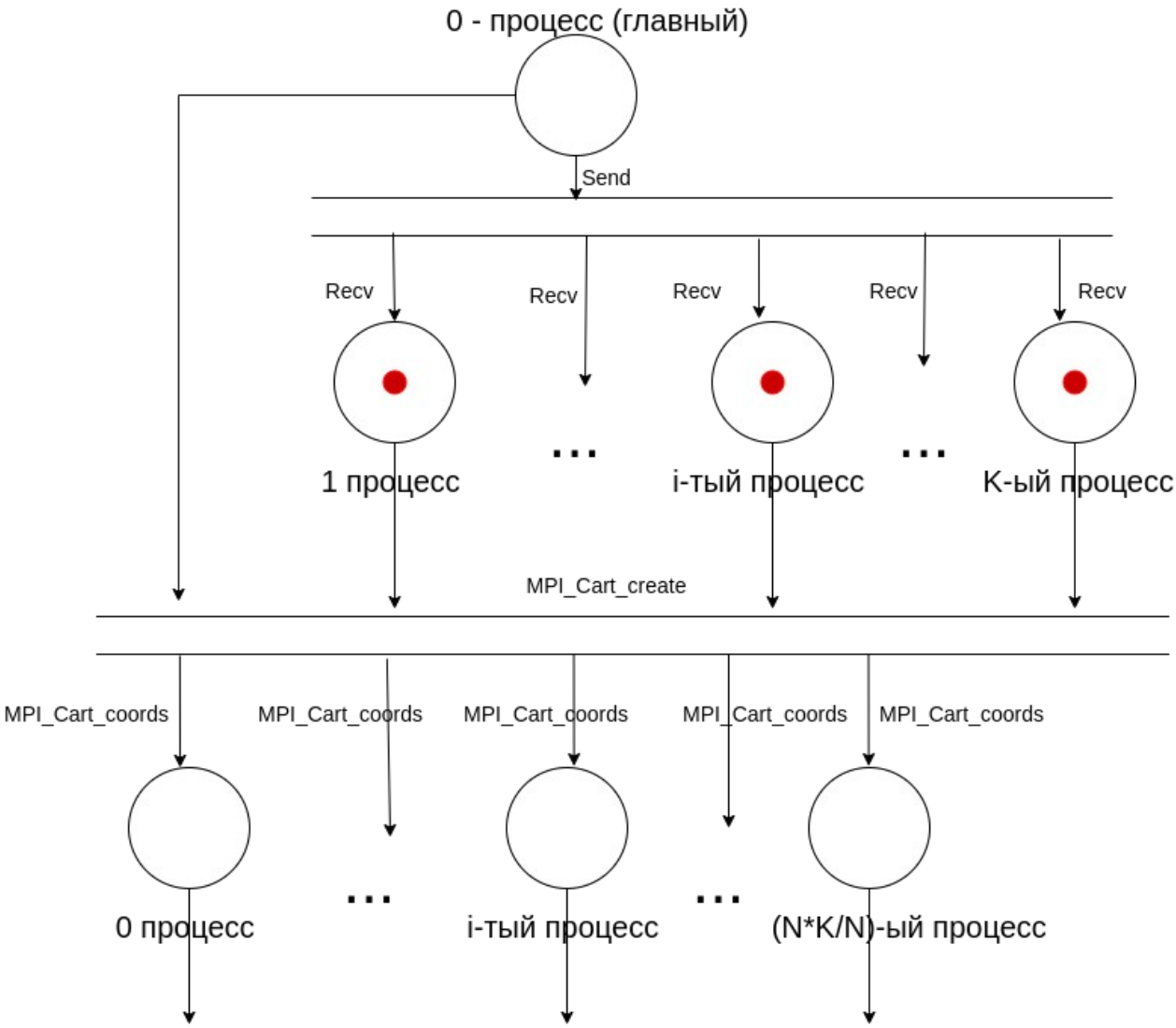
Формальное описание алгоритма с использованием аппарата Сетей Петри для n процессов:

K — количество процессов, N*K/N — количество процессов, включенных в декартову топологию.

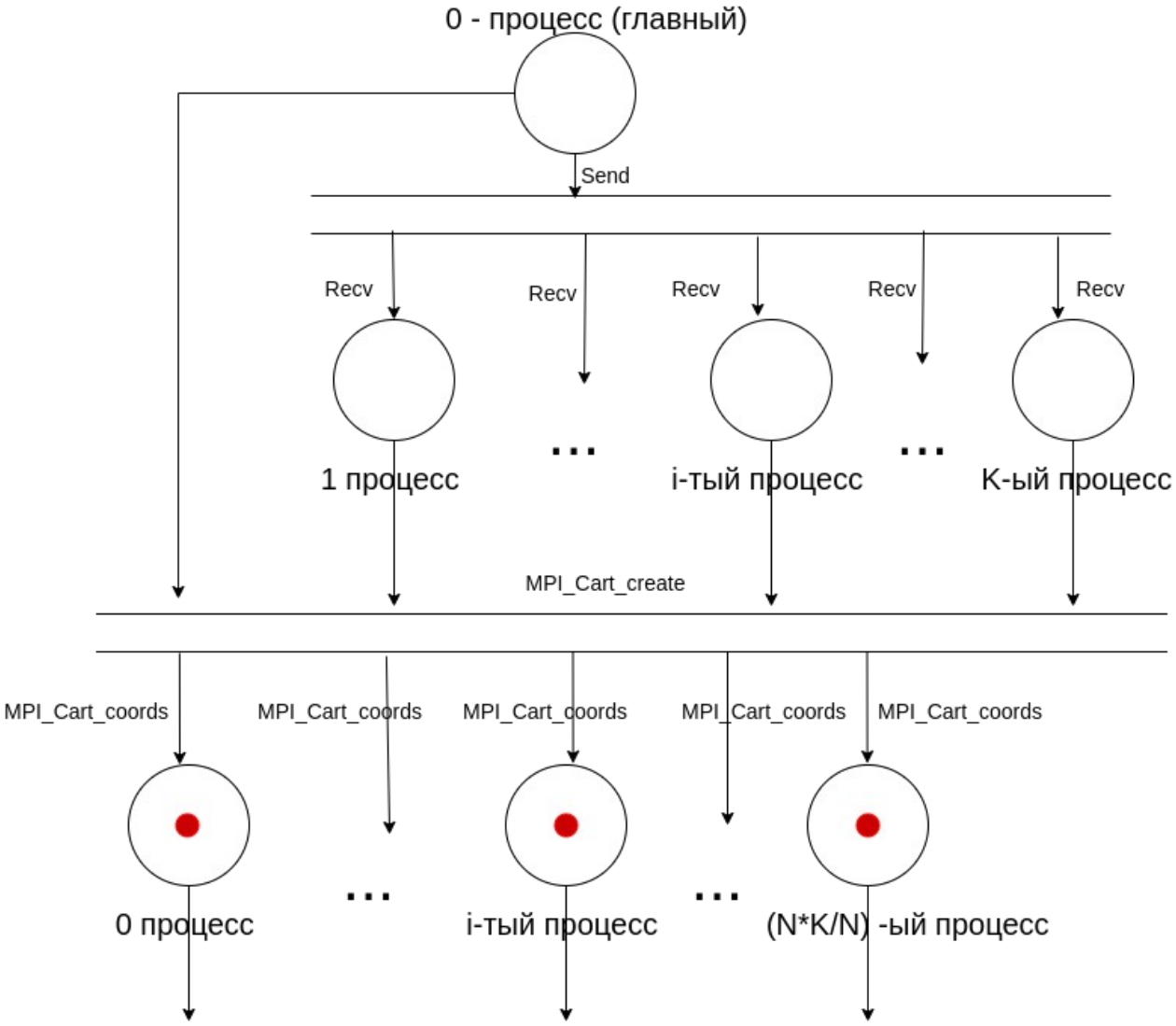
Шаг 1 — рассылаем из главного процесса значение N



Шаг 2: Не главные процессы принимают число N



Шаг 3: создание двумерной решетки и вывод координат процессов, входящих в нее



Листинг программы.

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

void main( int argc, char *argv[] ) {
   srand(time(NULL));
   int N = atoi(argv[1]);
   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;
    MPI_Status Status;

    //пересылка N всем процессам
    if ( ProcRank == 0 ){
        for (int i = 1; i<ProcNum; i++ ) {
            MPI_Recv(&N, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG,
MPI_COMM_WORLD, &Status);
        }
    }
    else
        MPI_Send(&N,1,MPI_INT,0,0, MPI_COMM_WORLD);

    int Dims[] = {N, round(ProcNum/N)}, Periods[] = {0, 0};
    MPI_Cart_create(MPI_COMM_WORLD, 2, Dims, Periods, 0, &Comm);
    int Coords[2];
    if(ProcRank < (N*round(ProcNum/N))){
        MPI_Cart_coords(Comm, ProcRank, 2, Coords);
        printf("Process = %d -> (%d; %d)\n", ProcRank, Coords[0], Coords[1]);
    }

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

Результаты работы программы

1) для 4 процессов и N = 3

```
white-lilac@white-lilac:~/Документы/ParAlg/lb5$ mpirun -np 4 --oversubscribe -quiet ./main.x 3
Process = 0 -> (0; 0)
Time: 0.000559
Process = 1 -> (1; 0)
Process = 2 -> (2; 0)
```

2) для 6 процессов и N = 3

```
white-lilac@white-lilac:~/Документы/ParAlg/lb5$ mpirun -np 6 --oversubscribe -quiet ./main.x 3
Process = 0 -> (0; 0)
Time: 0.000660
Process = 1 -> (0; 1)
Process = 2 -> (1; 0)
Process = 3 -> (1; 1)
Process = 4 -> (2; 0)
Process = 5 -> (2; 1)
```

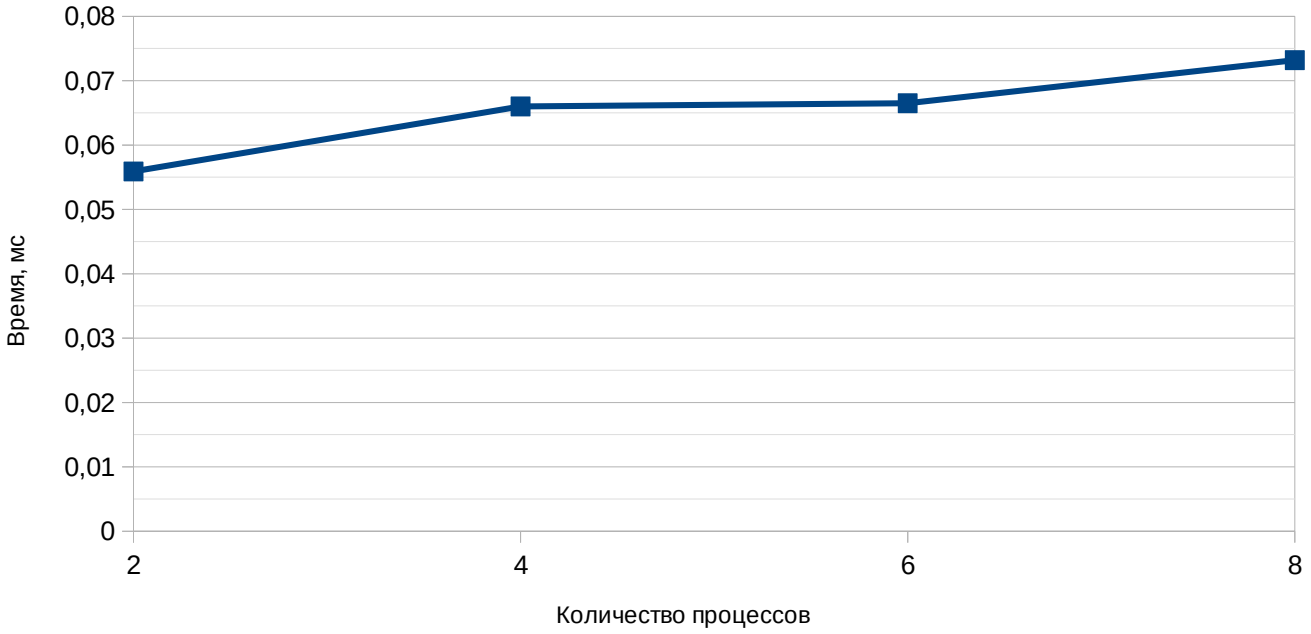
3) для 8 процессов и N = 3

```
white-lilac@white-lilac:~/Документы/ParAlg/lb5$ mpirun -np 8 --oversubscribe -quiet ./main.x 3
Process = 4 -> (2; 0)
Process = 5 -> (2; 1)
Process = 0 -> (0; 0)
Time: 0.000665
Process = 1 -> (0; 1)
Process = 2 -> (1; 0)
Process = 3 -> (1; 1)
```

4) для 10 процессов и N = 3

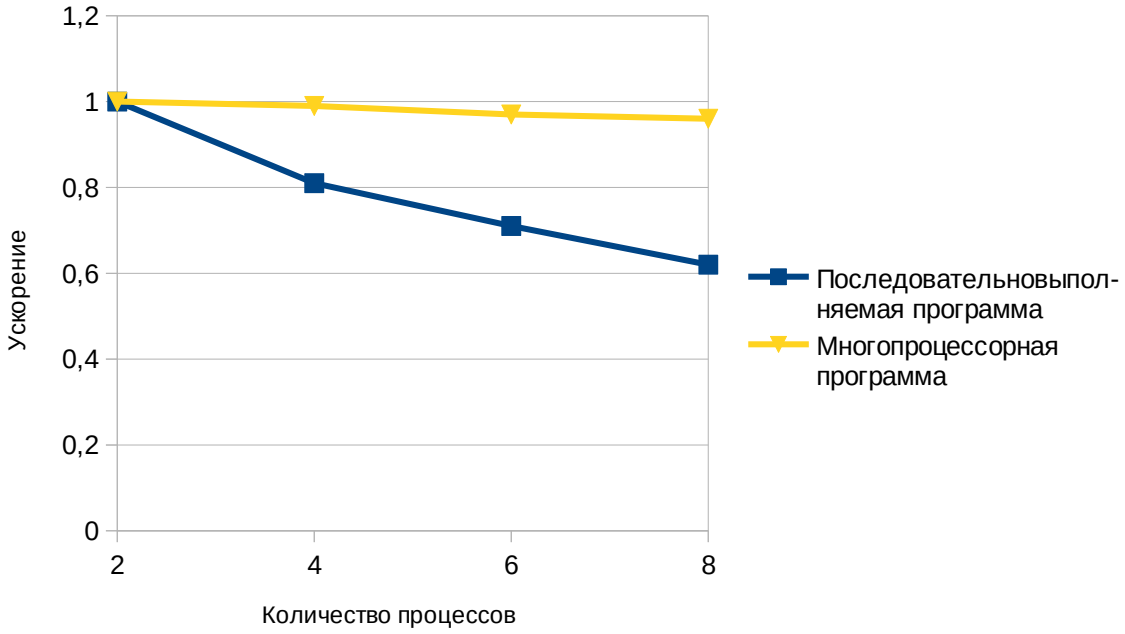
```
white-lilac@white-lilac:~/Документы/ParAlg/lb5$ mpirun -np 10 --oversubscribe -quiet ./main.x 3
Process = 0 -> (0; 0)
Time: 0.000732
Process = 1 -> (0; 1)
Process = 2 -> (0; 2)
Process = 3 -> (1; 0)
Process = 4 -> (1; 1)
Process = 5 -> (1; 2)
Process = 6 -> (2; 0)
Process = 7 -> (2; 1)
Process = 8 -> (2; 2)
```

График зависимости времени выполнения программы от количества процессов.



При увеличении количества процессов возрастает и количество процессов, которые нужно включить в двумерную решетку, следовательно возрастает и количество данных для вывода (координаты процессов в решетке), поэтому время выполнения программы незначительно возрастает (для удобства анализа во всех тестах было использовано число N=3). Заметим, что время возрастает на доли микросекунд, то есть увеличение количества процессов не сильно влияет на выполнение программы, так как они происходят параллельно.

График ускорения (эффективности):



Для однопроцессной 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 однопроцессной программы, где все действия выполняются последовательно.

Вывод.

Были изучены виртуальные топологии, а также была написана программа определения виртуальной топологии..